Escola Universitaria Politécnica

UNIVERSIDADE DA CORUÑA

Grado en Ingeniería Electrónica Industrial y Automática

TRABAJO DE FIN DE GRADO

TFG Nº: **770G01A175**

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

AUTOR: **MANUEL PALACIOS FRAGOSO**

TUTOR: **ESTEBAN JOVE PÉREZ**
HÉCTOR QUINTIÁN PARDO

FECHA: **JUNIO DE 2020**

Fdo.: EL AUTOR

Fdo.: EL TUTOR

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

ÍNDICE

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

Listado de Figuras

Listado de tablas

Listado de códigos de programación

Las siguientes librerías han sido empleadas como parte del programa final:

- OpenCV: Funciones de visión artificial.
- Scipy: Estadísticas con los datos.
- Numpy: Arrays, matrices y valores como pi.
- Time: Realizar pausas.
- Math: Cálculos.
- Picamera: Emplear la cámara en la Raspberry.

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

MEMORIA

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

Índice del documento MEMORIA

1 OBJETO

El objeto de este trabajo es implementar un sistema de reconocimiento y delimitación de superficies. Para ello, se buscará ser capaz de diferenciar entre los elementos básicos de toda habitación, como son paredes, suelo y techo.

Inicialmente, se hará un estudio del tipo de microcontrolador y software apropiado para aplicar las librerías OpenCV de reconocimiento de superficies. Posteriormente, tratará de implementarse el reconocimiento y delimitación de superficies en un entorno cerrado. Se estudiará la posibilidad de utilizar el reconocimiento de superficies tipo pared, suelo y techo para aplicar técnicas de realidad aumentada.

2 ALCANCE

El alcance de este proyecto se divide en las siguientes partes:

- Análisis del hardware y software adecuado para el uso de reconocimiento y delimitación de superficies.
- Aplicación y desarrollo de librerías OpenCV para el entorno seleccionado sobre diversas superficies tipo en un entorno cerrado.
- Estudio de la posibilidad de emplear el sistema de reconocimiento y delimitación de superficies en aplicaciones de realidad aumentada.

3 ANTECEDENTES

En la presente sección se expone el punto de partida y los condicionantes del proyecto. Este trabajo parte de la idea de ser capaz de llevar una habitación del mundo real al mundo virtual para poder trabajar con ella y, posteriormente, aplicar las modificaciones que se consideren oportunas con técnicas de realidad aumentada, cambiando así la forma en la que se percibe el entorno. La motivación se debe a las limitaciones que tienen a día de hoy los entornos en realidad virtual, ya que no permiten al usuario realizar desplazamientos en el mundo

real y se sigue dependiendo de un control remoto. Sin embargo, si se usase realidad aumentada para modificar el aspecto y todo lo observable en un cuarto, y se siguiesen percibiendo los obstáculos con técnicas de realidad aumentada, sería posible crear entornos en los que el usuario se desplazase por su habitación sin correr peligro con los objetos que le rodeasen. Por ello, este trabajo tiene como objetivo ser un primer paso capaz de dimensionar un cuarto, hallando sus principales límites. Serán necesarias técnicas de visión artificial para llevarlo a cabo.

3.1. Visión artificial

La visión artificial busca, mediante la toma, procesado y análisis de imágenes, proporcionar información del mundo real de forma que pueda ser tratada por una máquina. Se trata de dotar de “ojos” al sistema para que pueda analizar la información de un modo similar al que lo hace la vista humana.

Para el tratamiento de las imágenes, se descomponen en pequeñas unidades llamadas píxeles, que son la unidad básica de una imagen digitalizada en pantalla a base de puntos de color o en escala de grises. En la Figura 3.1 se dispone de un ejemplo de una imagen de una cebra en la que, ampliando las rayas de su costado, se puede comenzar a distinguir cada una de las pequeñas unidades cuadradas conocidas como píxeles.

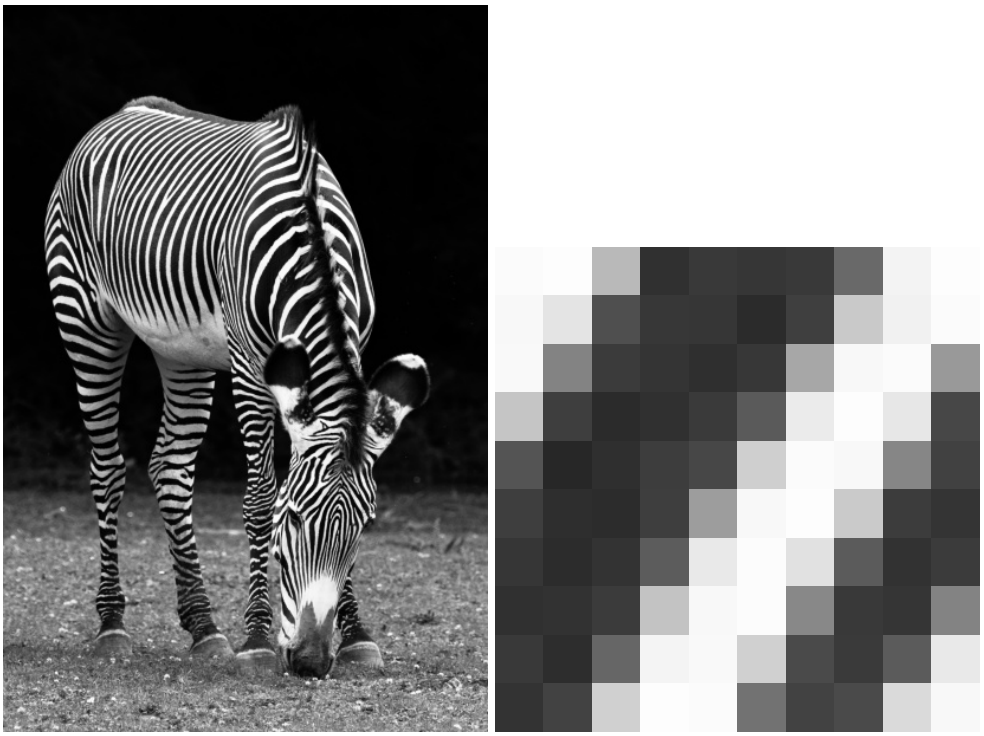
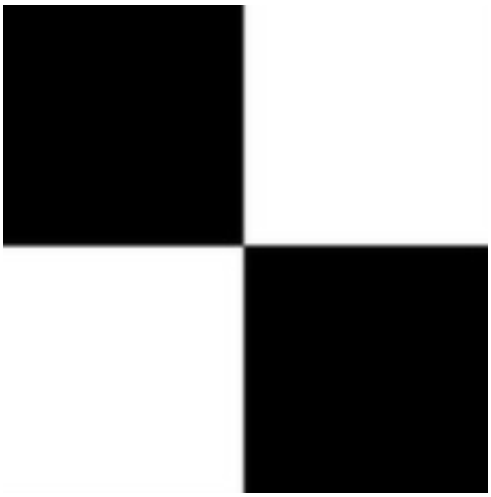


Figura 3.1 – Ejemplo de píxeles en una imagen

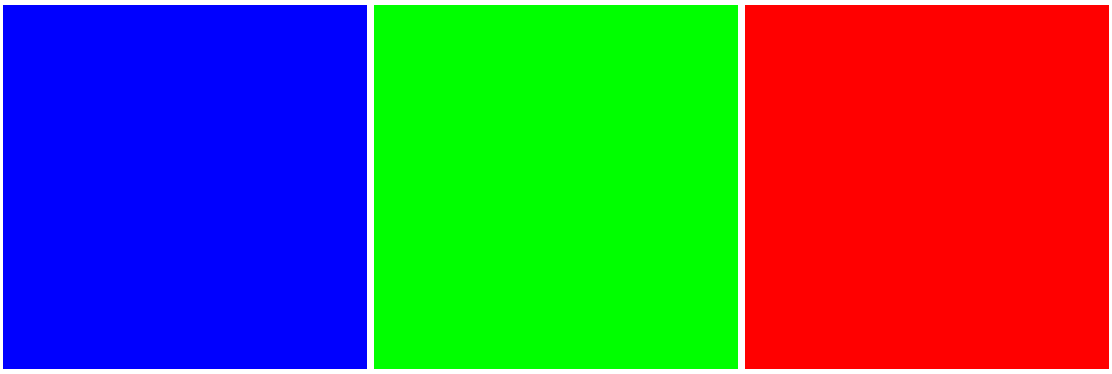
Los píxeles pueden contener distinta información dependiendo de si se trabaja en escala de grises o a color. Uno de los métodos más empleados cuando se trabaja a color es el RGB (en OpenCV es BGR), donde el color es en función de tres parámetros que corresponden a los

tres colores primarios en la luz: rojo, verde y azul. Al trabajar en escala de grises, cada píxel tiene asignado un único valor entre 0 y 255, siendo 0 para el color más oscuro y 255 para el más blanco posible (Figura 3.2), mientras que, si se trabaja a color, se dispone de un vector de longitud 3 mostrando los valores del píxel en BGR (Figura 3.3).



$$\begin{pmatrix} 0 & 255 \\ 255 & 0 \end{pmatrix}$$

Figura 3.2 – Ejemplo de cuatro píxeles en escala de grises y su matriz.



$$\begin{pmatrix} 255 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 255 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 255 \end{pmatrix}$$

Figura 3.3 – Colores primarios con su valor en BGR

En las técnicas de visión artificial, las imágenes son tomadas por una cámara que las transfiere para que sean procesadas en un ordenador y, en función de la aplicación realizada, el ordenador mandará instrucciones a un PLC, una máquina o mostrará el resultado obtenido. A menudo, la visión artificial es orientada al ámbito industrial, para el que se requerirá mayor solidez, fiabilidad y estabilidad comparado con una aplicación con un objetivo educativo como

es el caso. Algunos ejemplos de las aplicaciones de la visión artificial son la clasificación de productos, el manejo de brazos robóticos o la ejecución de labores de control calidad, por ejemplo, separando frutas recogidas que tengan alguna diferencia notoria en el color.

Entre las librerías de visión artificial, destaca OpenCV. Los filtros existentes en esta librería serán la base de este trabajo.

3.2. Ruido en imágenes

Al trabajar con imágenes no se puede evitar la existencia de interferencias. Estas, aparecen en la adquisición, la transmisión y el procesamiento de la imagen.

En el momento de ejecutar el programa, la imagen habrá sido adquirida por la cámara, transmitida al ordenador y procesada por este, por lo que resulta evidente que tendrá diferentes tipos de interferencias afectándole en mayor o menor medida. Se aplicarán soluciones sólo para las que parezcan causar problemas en la ejecución del programa.

Aunque existen multitud de tipos de ruido que pueden presentarse en una imagen, existen tres que son los más comunes:

- **Sal y pimienta:** Es un tipo de interferencia de impulso que suele distinguirse en las imágenes por la presencia de píxeles blancos y negros esparcidos en ella. Este tipo de ruido puede eliminarse rápidamente con el uso de filtros que tengan en cuenta el valor de los píxeles vecinos.

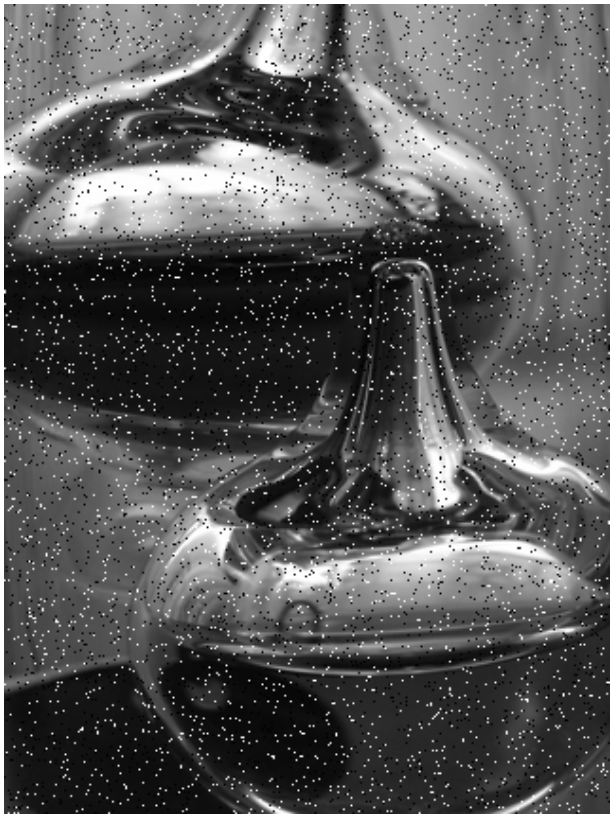


Figura 3.4 – Ruido sal y pimienta

- **Gaussiano:** Todos los píxeles de una imagen varían su valor siguiendo una distribución Gaussiana. Esto sucede, de acuerdo con el teorema central del límite, debido a que la unión de los diferentes tipos de ruido será cercana a una distribución gaussiana o normal, siempre y cuando el número de variables a sumar sea lo suficientemente grande.

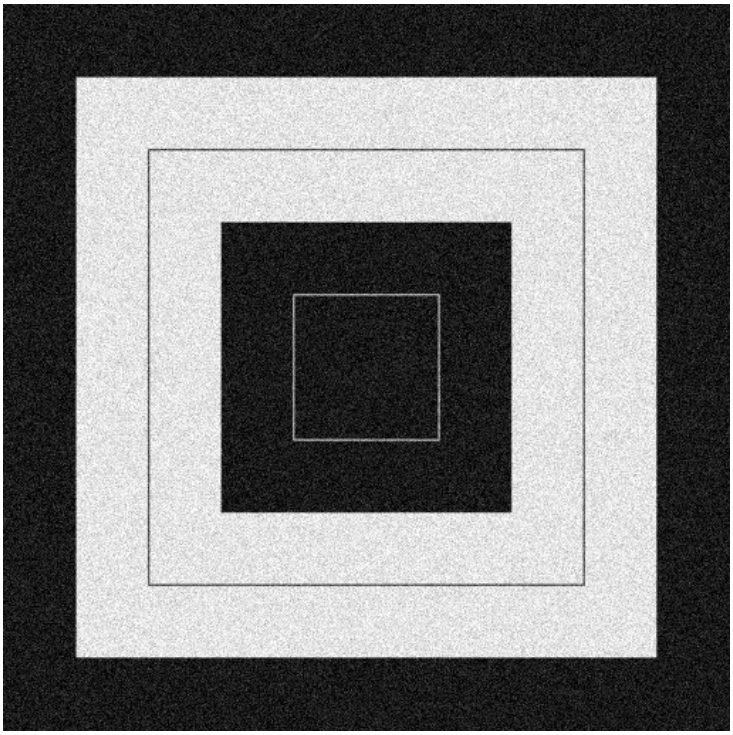


Figura 3.5 – Ruido gaussiano

- **Uniforme:** En el ruido uniforme, los píxeles afectados cambian su valor debido a una distribución uniforme, por lo que la imagen se verá como si estuviera ligeramente codificada. Para solventar este tipo de ruido, es necesaria la aplicación de un filtro de promedio espacial, aunque esto ocasionará una pérdida de calidad en la imagen. Como ejemplo, a la Figura 3.6 se le ha introducido ruido de este tipo en gran cantidad para que puedan observarse las diferencias con los anteriores y es que, en el caso de tratarse de imágenes a color, los píxeles alterados aleatoriamente adquieren un valor que no necesariamente estará suavizado conforme a una distribución determinada (como en el ruido gaussiano) ni será en blanco y negro (como en el ruido sal y pimienta).

Figura 3.6 – Ruido uniforme

3.3. Realidad aumentada

El término surgió cuando, en 1992, el científico Thomas P. Caudell, encargado de diseñar el Boeing 747, pensó que disponer de una pantalla que les fuese sirviendo de guía a los operarios para su ensamblaje sería de mucha ayuda [16].

La realidad aumentada busca la integración de contenidos gráficos en el mundo real. A menudo se utilizan móviles o gafas para la visualización. En 2012 fueron lanzadas al mundo las Google Glass, pero estas no tuvieron mucho éxito, fue en 2016, con el lanzamiento de Pokemon Go, cuando la realidad aumentada aumentó drásticamente su popularidad.

En el presente, la tecnología de realidad aumentada incluye:

- Sensores y dispositivos de entrada como GPS, cámaras, giroscopios o tecnología de reconocimiento de voz.
- Procesador y/o motor gráfico.
- Software que mediante técnicas de visión artificial mezcle el entorno real y el virtual.
- Display (opcional) desde el que visualizar el resultado de la unión del mundo virtual y el real. Este puede ir desde la pantalla de un ordenador hasta gafas en las que implementarlo.

Al hablar de una tecnología tan reciente como esta, no se puede evitar mencionar el gran futuro que tiene por delante y alguno de los proyectos más ambiciosos que pueden marcar el mundo de la electrónica.

La tendencia parece indicar que lentillas con realidad aumentada podrían volverse un accesorio común en las personas. Desde 2016, Samsung mantiene la patente de lentillas con cámara integrada, debido a esto, Apple comenzó a trabajar en productos más implicados en realidad aumentada y que, según varios medios de comunicación serán anunciados a lo largo del 2020. Al ser, ambas compañías, punteras en el mercado de la telefonía móvil, cualquier

movimiento que realicen en dirección a la realidad aumentada resulta de interés para el futuro de la electrónica.

Hasta hace poco se planteaba la recarga de los dispositivos como lentillas con cámara integrada mediante una antena pero, a principios de 2020, fueron anunciadas un prototipo de lentillas con una microbatería incorporada, lo que hace pensar que el gran salto de los dispositivos con realidad aumentada puede estar cerca [13].

4 NORMAS Y REFERENCIAS

4.1. Disposiciones legales y normas aplicadas

Se cumplirá la normativa establecida por la Escuela Universitaria Politécnica de Ferrol para la elaboración del TFG del Grado de Ingeniería Electrónica Industrial y Automática.

4.2. Bibliografía

[1] F. GIMÉNEZ-PALOMARES, J. A. MONSORIU, E. ALEMANY-MARTÍNEZ; *Modelling in Science Education and Learning*, Instituto Universitario de Matemática Pura y Aplicada Universitat Politècnica de València, (2016). DOI: 10.4995/msel.2016.4524.

4.3. Referencias web

[2] *Tema 1.- Introducción a la Visión Artificial*, Máster en Ingeniería Informática de la Universidad de Córdoba. [Fecha de consulta: 14 de marzo]. Disponible en: http://www.uco.es/users/ma1fegan/2015-2016/vision/Temas/ruido.pdf

[3] *Imagen 7.4 de Maulucioni, basado en un trabajo previo de Gabriela Ruellan.*, Wikipedia, the free encyclopedia. [Fecha de consulta: 10 de mayo]. Disponible en: https://es.wikipedia.org/wiki/Archivo:Amarillos.png

[4] *Image Filtering*. OpenCV documentation. [Fecha de consulta: 12 de abril]. Disponible en: https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#image-filtering

[5] *Imagen 3.4* . de Marko Meza, Wikipedia, the free encyclopedia. [Fecha de consulta: 10 de mayo]. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Noise_salt_and_pepper.png

[6] *Teorema del límite central*, Wikipedia, the free encyclopedia. [Fecha de consulta: 3 de abril]. Disponible en: https://es.wikipedia.org/wiki/Teorema_del_l%C3%ADmite_central

[7] *Deep Learning + Detección de bordes (HED) / Holistically-Nested Edge Detection*, Visión por computador de Carlos Julio Pardo. [Fecha de consulta: 17 de mayo]. Disponible en: https://carlosjuliopardoblog.wordpress.com/2019/04/17/\deep-learning-deteccion-de-bordes-hed-holistically-nested-edge-detection/

[8] *OpenCV: Canny edge detector*, OpenCV documentation. [Fecha de consulta: 5 de abril]. Disponible en: https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html

[9] *Imagen 3.5 del usuario DrWalencia de Wikipedia*, Wikipedia, the free encyclopedia. [Fecha de consulta: 10 de mayo]. Disponible en: https://es.wikipedia.org/wiki/Archivo:512x512-Gaussian-Noise.jpg

[10] *Arduino y Raspberry Pi dominan el hardware abierto... pero cada vez hay mejores alternativas*, BBVA API Market. [Fecha de consulta: 18 de marzo]. Disponible en: https://bbvaopen4u.com/es/actualidad/arduino-y-raspberry-pi-\dominan-el-hardware-abierto-pero-cada-vez-hay-mejores-alternativas

[11] *QUÉ ES LA VISIÓN ARTIFICIAL*, COGNEX. [Fecha de consulta: 18 de marzo]. Disponible en: https://www.cognex.com/es-es/what-is/machine-vision/what-is-machine-vision

[12] *La transformada de línea de Hough*, Unipython. [Fecha de consulta: 22 de marzo]. Disponible en: https://unipython.com/la-transformada-linea-hough/

[13] *Crean las primeras lentillas inteligentes autónomas: el paso previo para que nuestros ojos vivan en la realidad aumentada*, eldiario.es. [Fecha de consulta: 22 de mayo]. Disponible en: https://www.eldiario.es/tecnologia/Crean-lentillas-inteligentes-autonomas-aumentada_0_897660865.html

[14] *2.6. Manipulación y procesamiento de imágenes usando Numpy y Scipy*, GITHUB. [Fecha de consulta: 25 de marzo]. Disponible en: https://claudiovz.github.io/scipy-lecture-notes-ES/advanced/image_processing/index.html

[15] *Superponer imágenes usando la biblioteca de python*, AlwaysemMyhope. [Fecha de consulta: 15 de mayo]. Disponible en: https://alwaysemmyhope.com/es/python/704060-overlay-images-using-python-library-python.html

[16] *REALIDAD AUMENTADA. ¿QUÉ ES? CARACTERÍSTICAS Y TIPOS*, :IAT. [Fecha de consulta: 15 de mayo]. Disponible en: https://iat.es/tecnologias/realidad-aumentada/

4.4. Software utilizado

- Blender. Versión 2.81.
- Oracle VM Virtual Box.
- Raspbian OS (32-bits).
- Sublime Text. Versión 3.2.2.
- VNV Server y VNC Viewer.

5 DEFINICIONES Y ABREVIATURAS

- CPU: Central processing unit.
- GPU: Graphics processing unit.
- HD: High definition.
- HED: Holistically-nested edge detection.
- LCD: Liquid-crystal display
- RAM: Random access memory.
- ROM: Read-only memory.
- SD: Secure digital.

6 REQUISITOS DE DISEÑO

El sistema diseñado deberá cumplir los siguientes requisitos:

- Ser capaz de reconocer las paredes, el suelo y el techo de una habitación.
- Estar diseñado e implementado con software y hardware libre.
- Funcionar en la mayor variedad de condiciones posibles.

- Ser implementado con hardware de bajo coste.
- Disponer de una cámara para tomar fotografías en tiempo real y funcionar como sistema de realidad aumentada.

7 ANÁLISIS DE LAS SOLUCIONES

En este apartado se planteará un estudio de las diferentes alternativas de hardware y software para el presente proyecto, se analizarán dichas alternativas y se justificará la elección final.

7.1. Hardware

Una vez este finalizado el programa detector de superficies, será implementado en un dispositivo que permita su ejecución como una aplicación de realidad aumentada. Para probar su desempeño, se barajan varias alternativas de hardware de bajo coste.

- Arduino.
- Raspberry Pi.
- BeagleBone, MinnowBoard, etc.

7.1.1. Arduino

La placa Arduino es uno de los microcontroladores más famosos de la actualidad debido a ser de bajo coste, sencilla de utilizar y disponer de una gran cantidad de modelos con diferentes funcionalidades. Independientemente del modelo, al tratarse de un microcontrolador dispondrá, como mínimo, de una unidad central de procesamiento (CPU), unidades de memoria (RAM y ROM), puertos de salida y periféricos.

Los microcontroladores, por norma general, tienen velocidades de procesamiento mucho menores que los microprocesadores, por lo que resultan peor elección al trabajar con imágenes. Arduino sería mejor opción para realizar tareas sencillas en tiempo real, pero para tareas relacionadas con realidad aumentada será necesaria mayor capacidad de procesamiento.

Se dispone de una placa arduino UNO que podría ser empleada en el proyecto, sus características principales son:

- CPU: 16 MHz.
- Memoria SRAM: 2 KB.

- EEPROM: 1 KB.

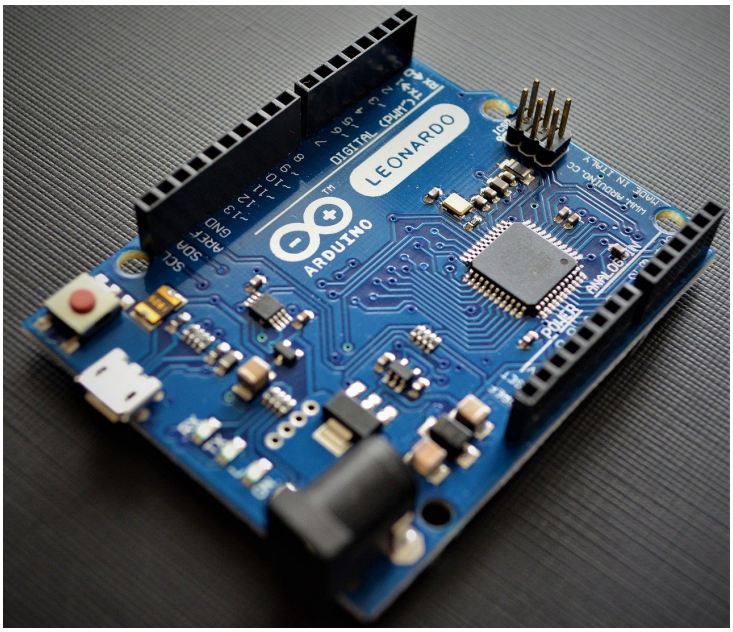


Figura 7.1 – Placa Arduino modelo Leonardo

7.1.2. Raspberry Pi

Raspberry Pi es una placa que dispone de un microprocesador, unidades de memoria y puertos. Todos los modelos de Raspberry Pi cumplen uno de los principales requisitos para su utilización en este trabajo, disponer de conector para cámara.

La presencia de un microprocesador permite a la placa alcanzar velocidades superiores a 1 GHz pero, será necesaria la utilización de un sistema operativo que lo controle. Para encargarse de este aspecto, Raspberry dispone de Raspbian, su propio sistema operativo, ligero y optimizado para sus placas, por lo que no resulta un inconveniente más allá de requerir unos conocimientos mínimos de Linux.

Para realizar el trabajo se dispone de una placa Raspberry Pi 3 modelo B, sus características principales son:

- CPU: 1.2GHz 64-bit quad-core ARMv8.
- GPU: Broadcom VideoCore IV que permite 1080p30.
- Memoria RAM: 1 GB (compartido con la GPU).
- Conexión Wifi.

Como unidad de almacenamiento es necesaria la utilización de una tarjeta SD. La presencia de conectividad Wifi posibilita el empleo de la placa a modo de dispositivo inalámbrico (siempre que se disponga de una fuente de energía portátil).

Figura 7.2 – Placa Raspberry Pi

7.1.3. Otras opciones

Existen fuertes competidores en el mercado, como pueden ser las placas de BeagleBone o las MinnowBoard. Ambas disponen de CPU y GPU, por lo que resultarían una opción válida para el trabajo, y, dependiendo del modelo de placa Raspberry, serían una opción mucho mejor. El problema de ambas opciones es que su precio es muy superior a Raspberry, llegando la MinnowBoard de Intel a superar incluso los cien euros.

7.2. Software

Una vez decidida la plataforma en la que será realizado el programa una vez esté finalizado, una Raspberry Pi 3 modelo B, habrá que decidir el software empleado:

- Sistema Operativo.
- Lenguaje de programación y librerías.
- Preparación del entorno.

7.2.1. Sistema Operativo

Para la realización del proyecto existen diferentes sistemas operativos que pueden ser utilizados pero, el sistema operativo seleccionado, deberá cumplir una serie de características que vienen determinadas por el hardware. Raspberry Pi 3 modelo B obliga a que el sistema operativo sea muy ligero, puesto que tan solo dispone de 1 GB de memoria RAM y está compartido con la GPU.

Existen distribuciones de Windows diseñadas para el IoT(Internet de las cosas) que son más ligeras y permiten su utilización en este tipo de hardware. En el último año, Windows se ha adpatado sacando una versión compatible, por lo que podría ser utilizado.

A pesar de todo, Raspbian, el sistema operativo propio de Raspberry Pi, ha sido optimizado para ser empleado en sus placas, por lo que su correcto funcionamiento está garantizado.

Además, al ser un sistema operativo en base Linux se trata de un sistema operativo libre y, por lo tanto, gratuito.

7.2.2. Lenguaje de programación y librerías

La selección del lenguaje de programación dependerá, en parte, de las librerías que vayan a ser empleadas para el código, por lo que se decide tratar ambas cuestiones simultáneamente.

Se plantean tres alternativas para ser empleadas como lenguaje de programación a lo largo del proyecto:

- **Matlab:** Es una gran alternativa para el tratamiento de imágenes pero tiene como gran desventaja no ser de código abierto, por lo que solo quien estuviese dispuesto a pagar una licencia podría ejecutar el código desarrollado en él. Existen alternativas libres como Octave, pero disponen de otra desventaja en común con Matlab, y es el peso del programa. Un sistema de hardware de bajo coste, como el que se está planteando, tendrá más problemas para la ejecución de este tipo de programas.

- **C:** Los últimos años ha ido aumentando su importancia en el campo de la visión artificial, por lo que a día de hoy es una opción sólida para la realización del proyecto. Debido a que es un lenguaje de programación que está aumentando su uso en este campo, cuenta con la desventaja de que, actualmente, es mucho más sencillo encontrar ayuda en internet para otros lenguajes que llevan más años controlando este campo, como por ejemplo Python o Java.

- **Python:** Junto con el lenguaje de programación Java, lleva años siendo empleado para aplicaciones de visión artificial. Es un lenguaje sencillo e intuitivo que dispone de gran cantidad de guías en internet, por lo que aprender lo básico de él no es una gran complicación.

En cantidad de documentación y ayuda en la red disponible, Python y C superan con creces a Matlab y, además, ambos, disponen de todas las librerías necesarias para realizar la aplicación de delimitación de superficies:

- OpenCV para las funciones de visión artificial.

- Scipy para poder realizar estadísticas con los datos.

- Numpy para poder trabajar con matrices y valores como pi.

- Time para poder realizar pausas.

- Math para poder realizar determinados cálculos.

- Picamera para poder emplear la cámara en la Raspberry.

Tras realizar búsquedas para documentarse sobre visión artificial con el empleo de OpenCV, se encuentra una mayor cantidad de ayuda disponible en Python.

Para escoger una librería de visión artificial, la decisión resulta más sencilla. OpenCV es una librería desarrollada por Intel que lleva activa más de veinte años, siendo considerada en la actualidad como la librería de visión artificial más popular del mundo. Gran parte de su popularidad estará basada en ser libre y estar disponible para multitud de plataformas, incluida Raspbian. Otra de las grandes ventajas de está librería y, que será consultada a menudo para la realización del proyecto, es la extensa guía de información de la que dispone en su página web referente a sus funciones [4].

7.3. Alternativas para el tratamiento inicial de las imágenes

Desde un primer momento, surgirán dos caminos a la hora de trabajar con imágenes:

- Trabajar en color: Se mantendría la imagen con los colores originales obtenidos por el programa y por lo tanto se tendrían tres datos de cada píxel (como se explica en el apartado 3.1: Visión artificial).
- Trabajar en escala de grises: Se dispondría solo de un valor por cada píxel (explicado también en el apartado 3.1: Visión artificial).

Figura 7.3 – Rueda de colores original y en escala de grises

Cuando se comparan los colores existentes al ser pasados a escala de grises, se encuentra que, colores como el violeta y el azul, resultan prácticamente indistinguibles al ojo humano. Esto puede resultar problemático cuando se busquen los límites si se da el caso de una habitación con dos superficies con colores difíciles de distinguir en dos paredes contiguas.

Puede parecer que eso obliga a trabajar con imágenes a color, para las que aumentará considerablemente la capacidad de procesamiento requerida, pero no es tan sencillo. Si bien en escala de grises algunos colores no se distinguen entre sí, lo mismo sucede si se mantienen los colores originales. Probando un filtro de OpenCV para dibujar límites se pueden encontrar

que en algunos casos en los que el filtro falla en escala de grises acierta cuando trabaja en color, pero también sucede a la inversa.

A simple vista se puede ver cómo, trabajando en color, la mayoría de colores se distinguen mejor entre ellos, pero cuando se trata de algunos tonos del mismo color, el cambio es más visible en la imagen convertida a escala de grises.

El filtro *Canny*, que será empleado como base del proyecto, realiza una detección de bordes en la imagen a partir de unos valores introducidos en él (explicado en profundidad en el apartado 8.5). Como demostración de que con ambos tratamientos iniciales se obtienen límites válidos pero variados, se establecen unos parámetros en el filtro *Canny* que lleven al error y se aplican por igual a la Figura 7.4, tanto en escala de grises como a color. Se obtiene que, trabajando inicialmente a color (Figura 7.5), detecta más límites pero, se queda sin detectar un límite que sí se detecta en escala de grises (Figura 7.6).

Es importante destacar que las Figuras 7.5 y 7.6 se tratan de un ejemplo con parámetros con poca capacidad de detección introducidos en el filtro *Canny*, para permitir así realizar una comparación. Con la mayoría de valores no habría problema para detectar todos los límites en una imagen tan sencilla, independientemente del tratamiento inicial de la imagen.

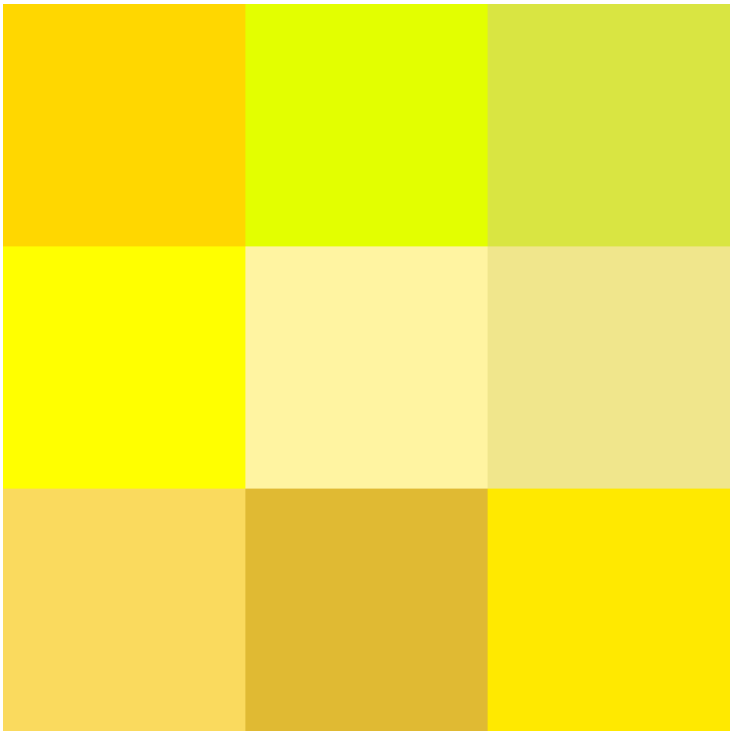


Figura 7.4 – Diferentes tonos de amarillo

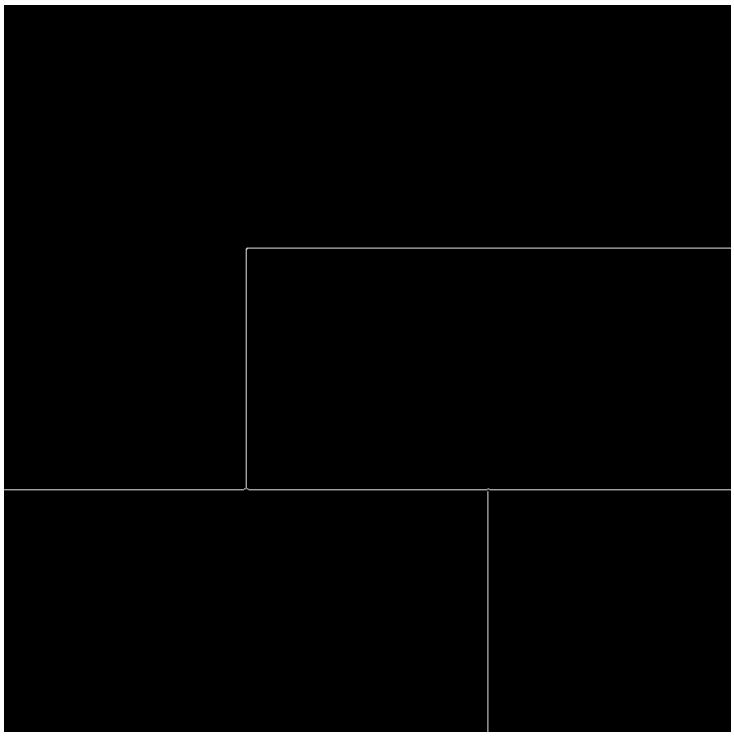


Figura 7.5 – Límites detectados en color

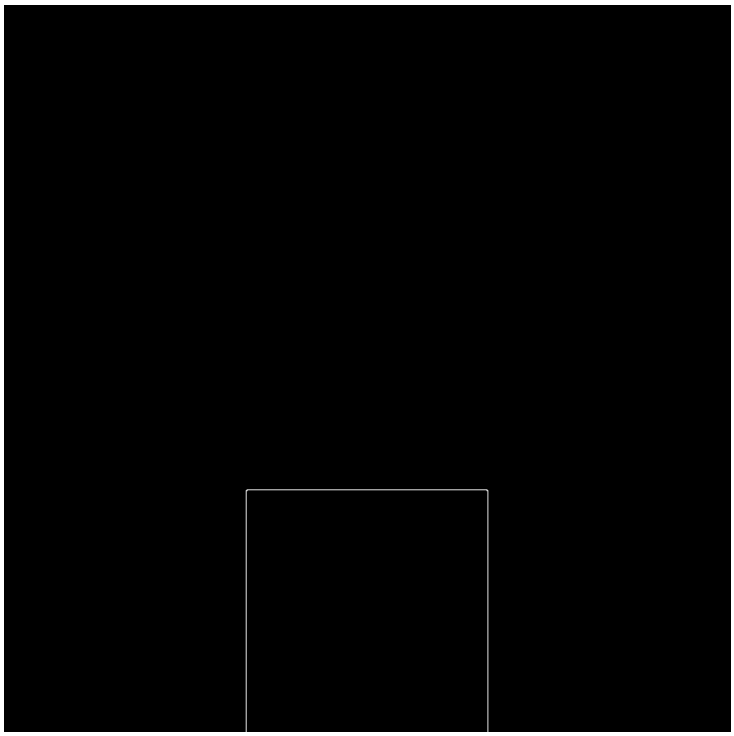


Figura 7.6 – Límites detectados en escala de grises.

La mejor opción si se obvia la velocidad del programa final y se piensa únicamente en la validez de los resultados, será trabajar con las imágenes en ambos formatos a la vez y utilizar las líneas de límites obtenidas en ambos casos. En la Figura 7.7 se tiene el resultado de la combinación de ambos métodos sobre la imagen inicial.

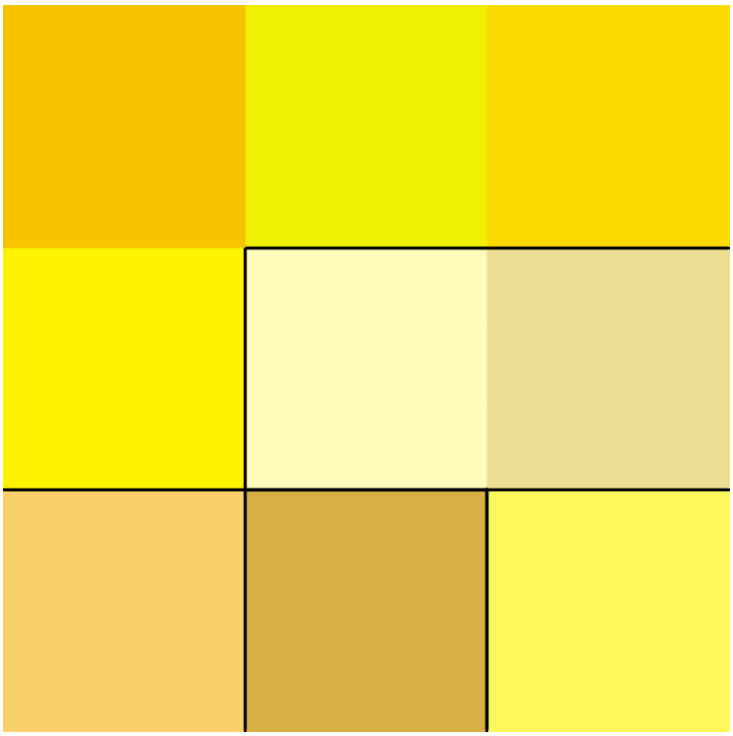


Figura 7.7 – Límites combinados sobre la imagen inicial

7.4. Alternativas para el tratamiento de ruido

Toda cámara o elemento de captura del mundo que nos rodea, introducirá siempre interferencias. En este caso se va a trabajar con diferentes cámaras fotográficas, por lo que habrá que preparar el programa para las interferencias producidas por cada una de ellas.

Será importante la presencia de filtros que eliminen los diferentes ruidos introducidos en cada imagen y permitan que un solo código sea capaz de trabajar con diferentes interferencias, para ello se emplearán filtros digitales.

Los filtros digitales podrán dividirse principalmente en filtros paso bajo, paso alto, direccionales y de detección de bordes. Los filtros de paso bajo suavizan la imagen y sirven para eliminar el ruido presente en ella, por lo que serán seleccionados para este proyecto. [1]

Los filtros paso bajo, también conocidos como filtros de suavizado, suavizarán las interferencias variando el valor de los píxeles en función de los que los rodean. En función del tipo de ruido existen filtros de suavizado más o menos apropiados para eliminarlo.

Siempre que se aplica un filtro de suavizado, provocará también una disminución de la calidad de la imagen y dificultará la detección del límite buscado. Lo ideal será usar el menor número de filtros posibles que permitan trabajar con la imagen sin imposibilitar la búsqueda de límites. Por ello, se comenzará el proyecto aplicando solo un filtro.

Destacan los siguientes tipos de filtros de suavizado:

- **Filtros de mediana:** Cogen el valor de un píxel y lo reemplazan por la mediana de los valores de los píxeles que lo rodean. Este tipo de filtros funcionan muy bien con el ruido sal y pimienta pero, en un principio, este ruido no resultará una preocupación, por lo que

se descarta su uso.

- **Filtros de media:** Solo varían respecto a los de mediana en que, en lugar de sustituir el valor del píxel por el valor de la mediana, lo hará por el de la media.

- **Filtros gaussianos:** Resultarán mejor opción para eliminar el ruido de la imagen de una forma más genérica. El motivo de su uso es que la máscara que crean para eliminar/-suavizar el ruido responde a una función gaussiana y, como según el teorema central del límite, el ruido responderá a una distribución gaussiana, este filtro parece la mejor opción para eliminarlo sin afectar en exceso a los bordes.

Si se logra que el sistema genere los mismos resultados con ruido y sin él, se tratará de un sistema robusto.

7.5. Alternativas para la detección de límites

Una vez eliminado, en la medida de lo posible, el ruido presente en la imagen, el programa tratará de detectar una gran cantidad de bordes y quedarse con los más importantes para el proyecto. Para esto último, se plantearán en esta sección, dos posibilidades iniciales de trabajo:

- Detección de esquinas.

- Detección directa de bordes.

Previamente, para la detección de gran parte de los bordes existentes en la imagen, se emplea el filtro *Canny*, que tiene como inconveniente que las líneas que marca no pueden ser seleccionadas. Por ello, partiendo de esa base, se estudian distintas alternativas para la detección de los bordes.

A día de hoy, ha surgido algún filtro como el HED, que parece proporcionar mejores resultados que el filtro *Canny* para detectar bordes. Sin embargo, se descarta su prueba y su uso, puesto que necesita mucha mayor capacidad de procesamiento, al punto de no bastar con una CPU para implementarlo y requerir un sistema con una GPU potente. Tiene el inconveniente a mayores de ser incompatible con las gráficas NVidia, por lo que resultaría imposible probarlo con el ordenador disponible. [8]

7.5.1. Detección de esquinas

Existen diferentes filtros en OpenCV que permiten detectar esquinas en los objetos, por ello surge la posibilidad detectar conjuntos de puntos cercanos entre si que puedan indicar la presencia de un determinado objeto. Para lograr esto, sería necesario la creación de clústers con los conjuntos de datos.

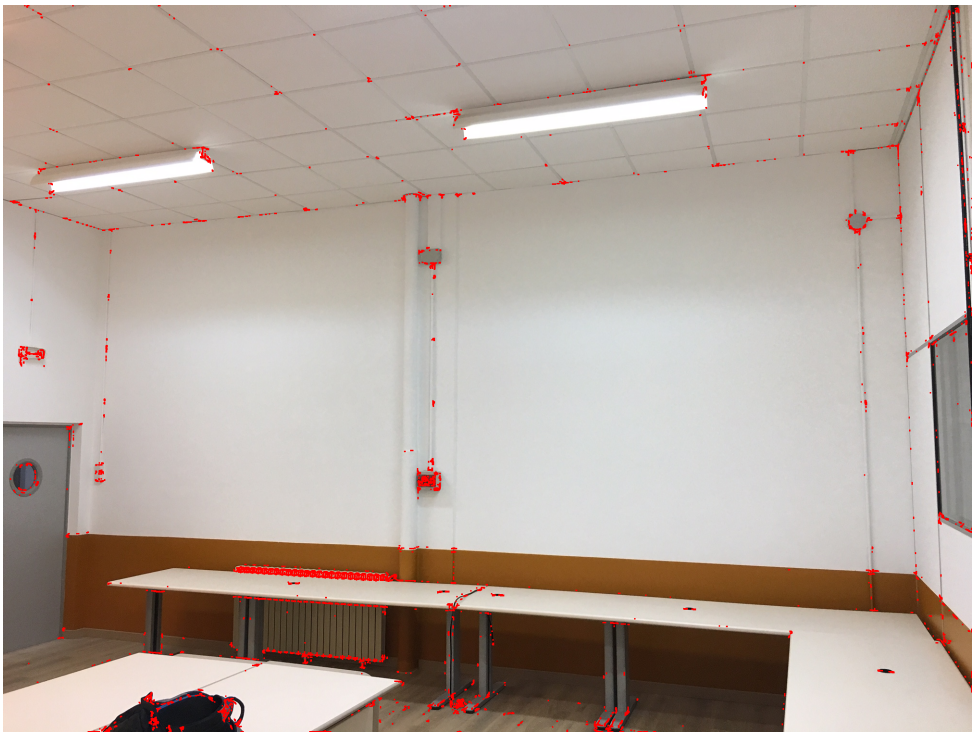


Figura 7.8 – Laboratorio con filtro Harris

Sin embargo, surgen diversos problemas con esta idea. El primero es que, mediante este método, el ruido sal y pimienta y todos los demás ruidos de tipo impulso pueden entorpecer el objetivo. Podría tratar de solventarse, pero no sin el riesgo que conlleva perder calidad en la imagen y, con ellos, los valores que representan esquinas útiles.

El segundo problema llega al tener que aumentar en gran cantidad los datos con los que trabajar, puesto que cualquier pequeño objeto presente en la habitación genera mínimo la detección de una esquina. En primera estancia, puede parecer que esta esquina no supondría un problema, pero filtros como *CornerHarris* de OpenCV solicitan como dato de entrada el número máximo de puntos que se quieren detectar. Si no fijas los suficientes puntos perderías de forma aleatoria otros puntos importantes para detectar los bordes. Además, mayor cantidad de puntos aumentaría la dificultad para que los clúster trabajasen correctamente.

El tercer problema, y el más importante, es que si obvias la idea de los clúster e intentas unir puntos para formar líneas, cualquier esquina de un objeto puede dar líneas de límites erróneos. Por suerte, ya existen filtros que buscarán trazar las líneas de los bordes directamente.

Se comprobó también con el filtro *GoodFeaturestoTrack* pero los resultados fueron similares.

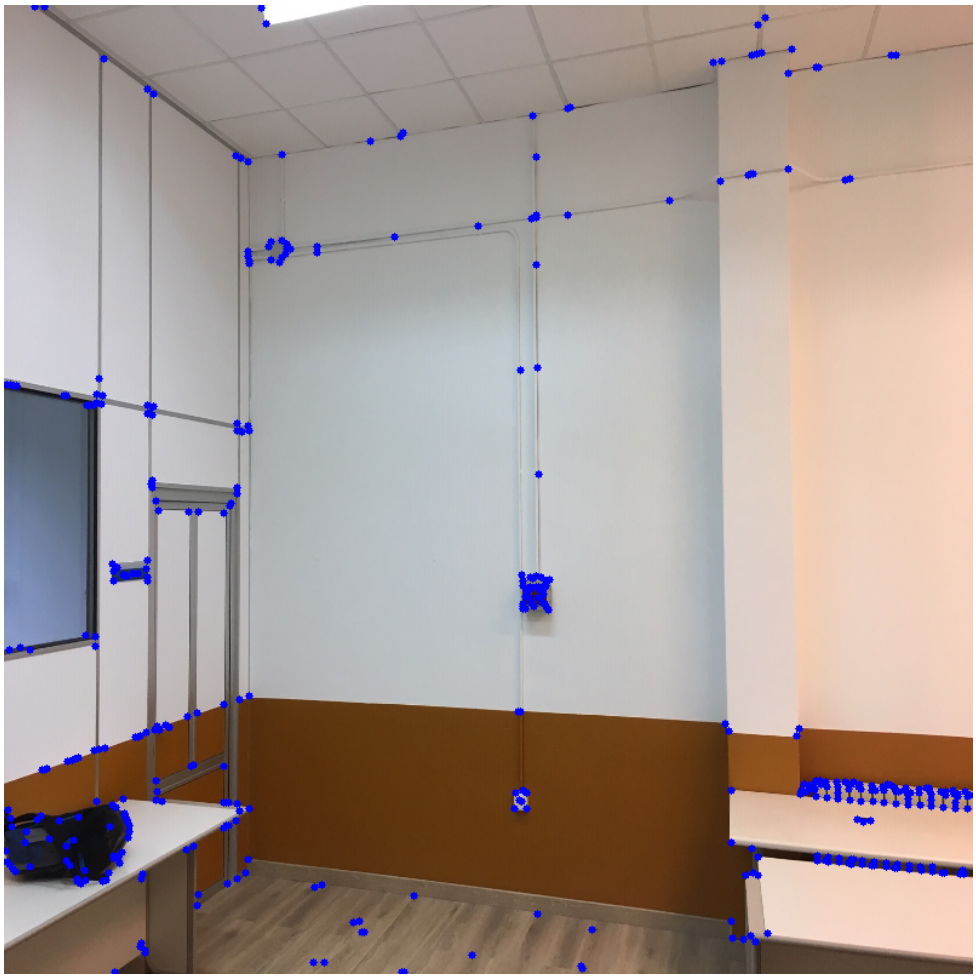


Figura 7.9 – Laboratorio con Goodfeatures

7.5.2. Detección directa de bordes

Existen diversos filtros que buscan directamente detectar los bordes entre objetos en una imagen. *HoughlinesP* entrega como salida una matriz con las líneas detectadas, permitiendo trabajar con los datos con cierta comodidad. Aunque no tenga tantos problemas como los filtros de detección de esquinas, las luces y los objetos también le generan dificultades y líneas falsas al buscar los bordes.

Para la detección de n líneas, la matriz que devolvería la función sería así:

$$\begin{pmatrix} x_1 1 & y_1 1 & x_1 2 & y_1 2 \\ x_2 1 & y_2 1 & x_2 2 & y_2 2 \\ ... & ... & ... & ... \\ x_n 1 & y_n 1 & x_n 2 & y_n 2 \end{pmatrix}$$

El gran obstáculo de este trabajo será establecer unos parámetros iniciales que permitan disponer de todas las líneas de bordes importantes e ignorar correctamente la mayor cantidad de líneas erróneas posibles.

Existe una variante para encontrar límites que, punto a punto, va detectando los contornos

presentes en la imagen. Para detectar un objeto sobre un fondo de otro color es muy útil pero, al aplicarlo a una habitación (Figura 7.10), su comportamiento es erróneo y muchas veces impredecible. Basta con fijarse que en la pared del fondo se guía por la sombra, mientras que en la mesa de la zona inferior izquierda la ignora perfectamente.

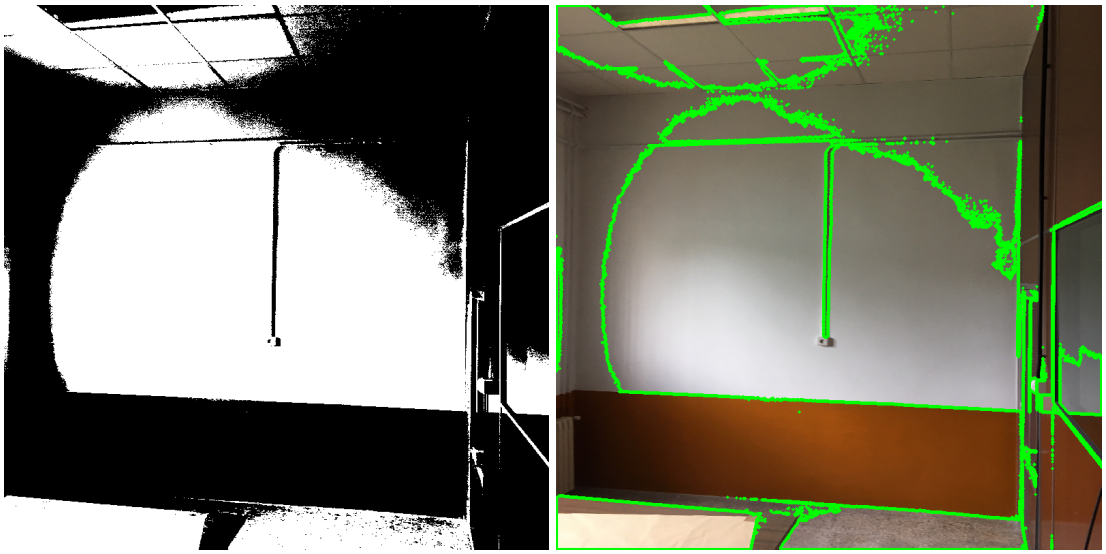


Figura 7.10 – Dibujado de contornos en laboratorio 2

7.6. Soluciones a los problemas de la realidad aumentada

Al realizar una aplicación de realidad aumentada surgirán problemas debido a que la realidad tiene tres dimensiones y la imagen capturada solo dos. Esto genera el principal problema de este tipo de aplicaciones, la profundidad.

Para ser capaz de detectar la profundidad cuando se pasa de un entorno real a uno virtual, existen dos opciones:

- Disponer de dos cámaras que proporcionen una imagen de la misma habitación desde diferentes perspectivas en un mismo instante de tiempo.
- Disponer de dos fotografías obtenidas desde distintas posiciones en distintos momentos de tiempo. Para esto, será necesario que todos elementos fotografiados permanezcan en la misma ubicación.

El problema de la profundidad se solucionaría con fotografías adquiridas de los dos métodos previamente mencionados y realizando los siguientes pasos:

1. Recopilación de imágenes de entrenamiento de multitud de cuartos desde distintos ángulos.
2. Mediante deep learning, entrenar redes neuronales para realizar mapas de profundidad de las habitaciones.
3. Emplear el aprendizaje de la red neuronal en la creación de mapas de profundidad de las habitaciones que vayan a ser dimensionadas.

Resulta casi imposible obtener suficiente material libre para entrenar una red y realizar mapas de profundidad de las habitaciones. Esta técnica es empleada, por ejemplo, por Google para las aplicaciones de realidad aumentada.

Como no será viable captar la profundidad de tal manera, se buscará lidiar con ella de forma más sencilla a partir de dos problemas:

- Perspectiva.
- Oclusión.

Fijando una perspectiva aproximada de la fotografía tomada, se podrá trabajar con unas condiciones estáticas y conocer la ubicación aproximada de los límites buscados. Para este proyecto, se tomará la cámara como si estuviese ubicada a media altura en el punto central de una habitación y enfocando en dirección a una de las paredes. El programa será realizado todo lo flexible en la ubicación de la cámara que sea posible permitiendo así un margen de error en su ubicación muy elevado.

Debido a la oclusión, se dificultará reconocer cuál es la línea del límite buscado respecto a los bordes de otros objetos presentes, para tratar con este problema se trabajará con la repetición de pendientes y proyecciones de líneas. En la Figura 7.11, se puede observar cómo por la presencia de una caja queda oculto el límite entre una mesa y una pared, dificultando así su localización y haciendo que el programa pueda confundir el límite caja-pared con el límite mesa-pared.

Figura 7.11 – Ejemplo de oclusión

Entre ambos problemas, perspectiva y oclusión, el de la oclusión es el que resulta más complicado, puesto que en función de condiciones variables como la iluminación o la simple presencia de un objeto en la sala, los parámetros iniciales pueden precisar ser cambiados. Por ello, este proyecto busca que los parámetros iniciales pasen a depender en la mayor medida posible solo de la cámara que tome la imagen.

Si se trata de realizar una aplicación de realidad aumentada que dimensione todo un cuarto a partir de fotografías de cada una de las paredes, será importante recordar que todos los elementos deben permanecer en el mismo lugar durante la toma de las fotografías, o podrían dar lugar a error.

8 RESULTADOS FINALES

En esta sección se explicará la solución final alcanzada y cada uno de los pasos realizados en ella.

8.1. Esquema general del programa

Para facilitar la comprensión del programa diseñado, en este apartado se indicarán los pasos ejecutados por él, una breve descripción de cada uno y la sección en la que están explicados en mayor profundidad.

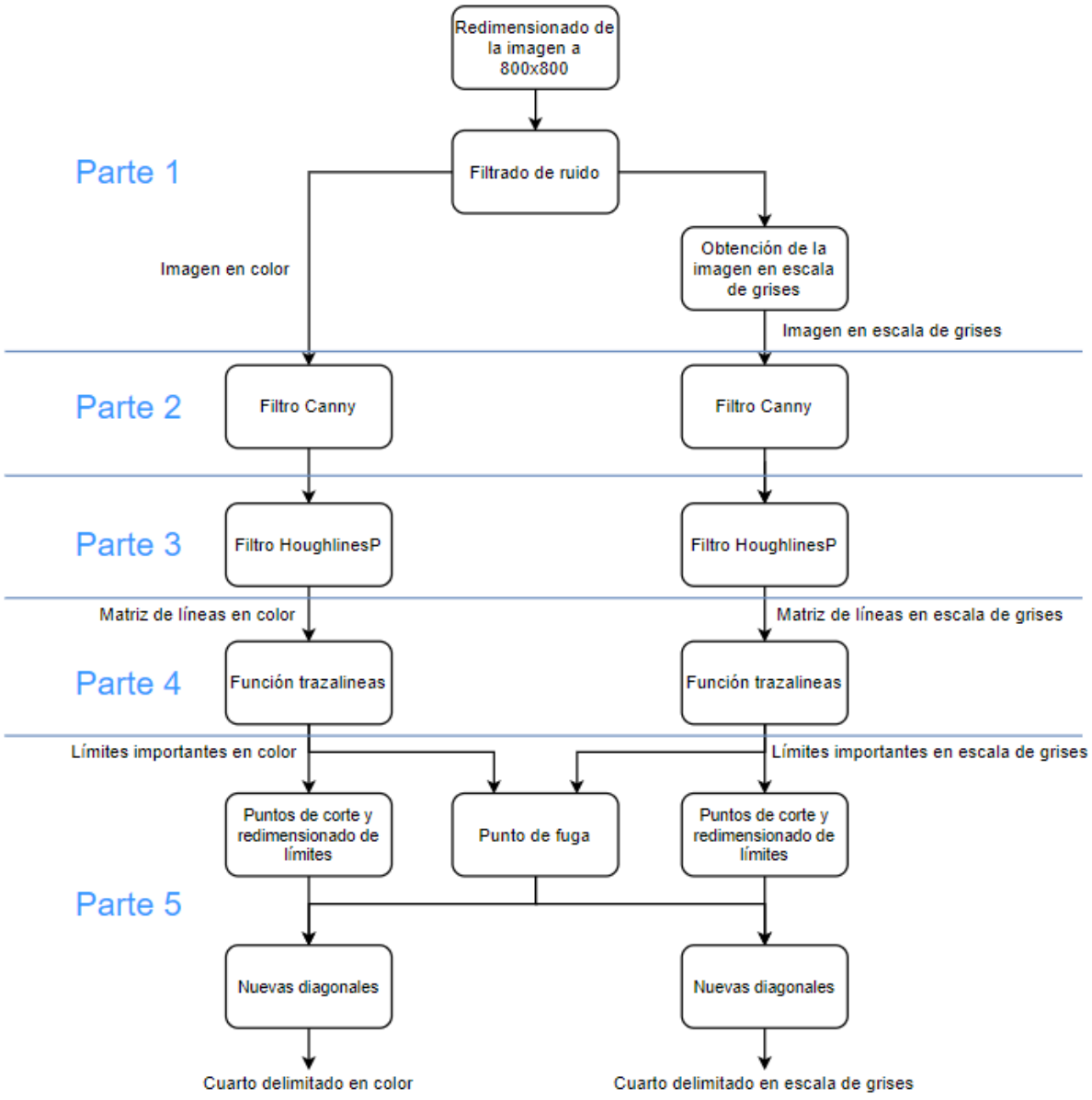


Figura 8.1 – Esquema general del programa final de delimitación de superficies

Mediante líneas azules en la Figura 8.1, se divide el programa de detección de límites en cinco partes:

- **Parte 1:** Se prepara la imagen para poder buscar los límites presentes en ella. Primero, se escala para trabajar siempre con imágenes del mismo tamaño y después, debido a las interferencias que puedan haber afectado a la fotografía, se filtra el ruido para evitar problemas en la detección de bordes. Por último, se convierte la imagen en escala de grises para trabajar con ella tanto en color como convertida. Apartado 8.4: Tratamiento inicial de imágenes.
- **Parte 2:** Se realiza una detección de bordes mediante un filtro *Canny* que marcará, en blanco sobre un fondo negro, todos los contornos presentes. Apartado 8.5: Filtro *Canny*.
- **Parte 3:** Se aplica el filtro *HoughlinesP* para detectar líneas en bordes previamente obtenidos en la Parte 3. Se obtiene así el punto inicial y final de las líneas rectas halladas para poder trabajar con ellas. Apartado 8.6: Filtro *HoughlinesP*.
- **Parte 4:** Se diseña y se aplica una función que, a partir de un grupo de líneas, seleccione aquellas que tienen la pendiente más repetida y, de estas, aquellas que tienen una proyección más repetida en el borde de la imagen. Apartado 8.7: Función trazalineas.
- **Parte 5:** Se calcula el punto de fuga en función de las diagonales disponibles. Se hallan también los puntos de corte entre el resto de límites y se redimensionan para que coincidan con ellos. Usando el punto de fuga y los puntos de corte, se trazan nuevamente las diagonales de la habitación. Apartado 8.8: Punto de fuga y redimensionado.

Tras haber delimitado el cuarto con el que se esté trabajando, se procederá a reemplazar la superficie presente en la pared central (Apartado 8.9: Superposición de una imagen). Realizando cíclicamente una toma de fotografías y la ejecución del programa diseñado, se realiza una sencilla aplicación de realidad aumentada (Apartado 8.10: Aplicación de realidad aumentada).

8.2. Preparación del entorno

Al haber seleccionado Raspbian, el trabajo será preparado para ser ejecutado en un sistema operativo basado en Linux y, por lo tanto, habrá que instalar una serie de librerías desde el terminal por medio de la instrucción “sudo” para actuar como el superusuario.

La mayoría de librerías se podrán instalar con la instrucción “sudo apt-get install” o mediante la instrucción “pip3 install” en caso de programar con python 3, si no se excluiría el 3 de la instrucción “pip”.

Una vez instalados todos los componentes y asegurando que no haya problemas de incompatibilidad entre las versiones de las librerías, podrá comenzarse a programar.

Antes de ser probado en la placa real, será instalado el mismo sistema operativo en la máquina virtual Oracle VM VirtualBox, así se dispondrá de mayor comodidad durante la programación y se evitará depender de una red Wifi para trabajar.

Cuando el código esté listo para ser ejecutado desde la Raspberry Pi, se instalará el programa VNC Server en ella y se configurará como programa de inicio. En el ordenador, desde donde se controlará la Raspberry de forma remota para ahorrar la necesidad de periféricos adicionales, se instalará VNC Viewer para poder monitorizar y actuar sobre el sistema mediante una conexión Wifi.

8.3. Toma de imágenes

Lo primero será disponer de imágenes en las que ir probando el proyecto y las diferentes alternativas estudiadas. En la vida real es difícil disponer de cuartos vacíos sin objetos que dificulten la búsqueda de los límites entre superficies por lo que habrá que buscar una alternativa mediante un programa de modelado.

Para realizar el proyecto se dispondrán de cuatro opciones para probar su efectividad:

- **Habitación simulada:** Se creará un cuarto simple, utilizando el programa de modelado Blender, en el que se pueda jugar con los colores de las paredes y con la iluminación. Así se podrá obtener la certeza de que el programa detecta bien los límites y que en caso de fallar se debería a la interferencia de objetos o ruido introducido por el mundo real.

- **Fotografías del laboratorio y su sala adyacente:** Se obtienen de la Escuela Universitaria Politécnica de Serantes empleando la cámara incorporada en un teléfono móvil. Al trabajar con imágenes de mayor calidad, la cantidad de datos manejados también aumentará y el programa será más lento.

- **Fotos de libre uso del repositorio de Pixabay:** Estarán tomadas con diferentes cámaras y podrán tener aplicados diferentes filtros previos, por lo que dificultarán la detección y deberán llevar el programa al límite de su funcionamiento o directamente al error.

- **Imágenes de la prueba real del dispositivo de realidad aumentada:** Al implementar el sistema de realidad aumentada en la Raspberry, las imágenes serán tomadas nuevamente por una cámara distinta. Probando el dispositivo final en el mismo laboratorio donde fueron obtenidas las fotografías con el móvil, podrían compararse los resultados y comprobarse así la influencia de la variación de la cámara fotográfica. Esta comprobación no será posible debido al confinamiento por el COVID-19.

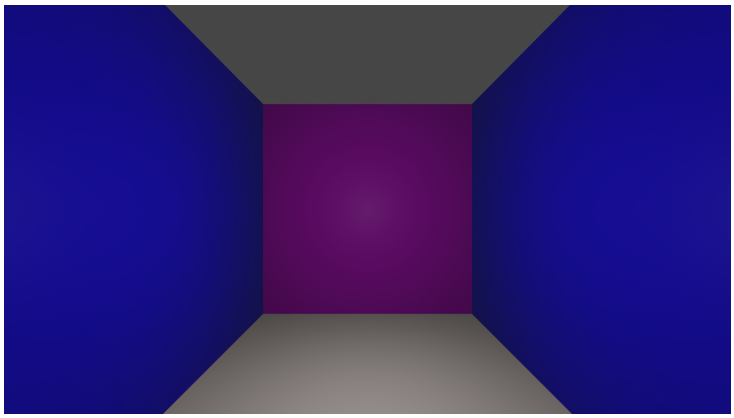


Figura 8.2 – Cuarto básico con iluminación central

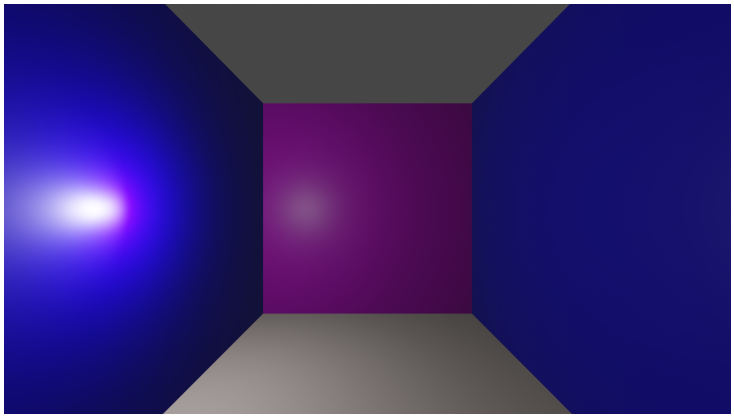


Figura 8.3 – Cuarto básico con iluminación lateral

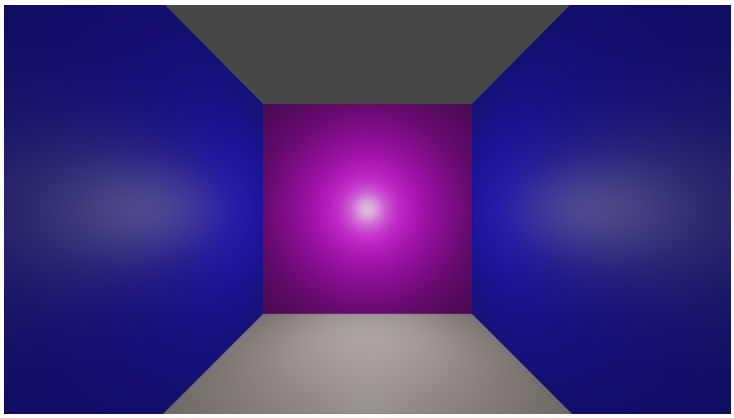


Figura 8.4 – Cuarto básico con iluminación fuerte

Añadir algún objeto en el cuarto simulado, permitirá aislar el caso en el que la imagen no tenga ruido por su adquisición y transmisión, y ver cómo le afecta la combinación de luces, sombras y la presencia de un objeto a la hora de delimitar las superficies.

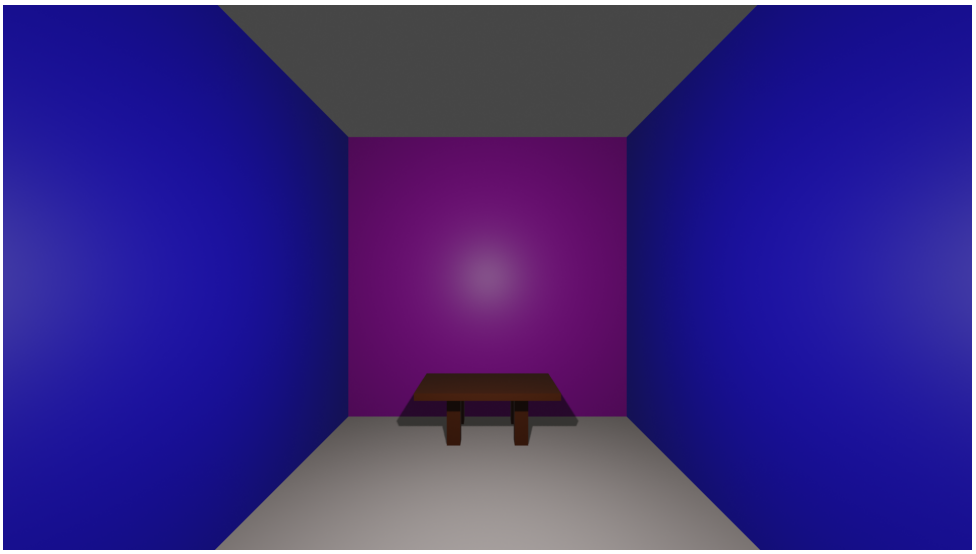


Figura 8.5 – Cuarto con mesa e iluminación central

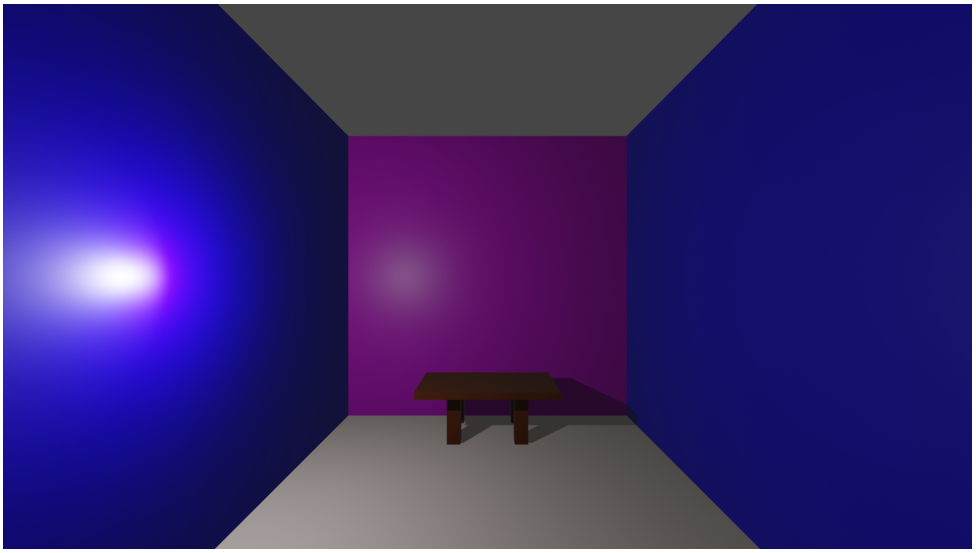


Figura 8.6 – Cuarto con mesa e iluminación lateral

Las imágenes del laboratorio aportarán la opción de, una vez comprobado el resultado en un entorno simulado con una mesa, ver cómo afecta el ruido al programa. Habrá que estudiar cómo reducir la interferencia que este produce para que el programa se ejecute de forma correcta al usar cámaras reales. Además, surgirá el problema de las texturas reales de los elementos, que añadirán mayor dificultad, y de nuevos objetos paralelos a los límites buscados, como las tuberías que recorren las paredes.

Figura 8.7 – Laboratorio 1

Figura 8.8 – Laboratorio 2

Figura 8.9 – Laboratorio 3

Por último, se buscará verificar el funcionamiento del programa con mayor variedad de casos y ver cómo reacciona cuando la cámara de procedencia de las imágenes o su resolución es variada. En el caso de funcionar en un porcentaje alto de estas imágenes, se tendría un programa plenamente funcional para el mundo real.

A mayores, se estudiará la posibilidad del montaje de un dispositivo de realidad aumentada y se trabajará con las imágenes obtenidas directamente por la cámara de este.

8.4. Tratamiento inicial de las imágenes

Lo primero que se hará una vez leída la imagen, será redimensionarla para trabajar siempre con el mismo tamaño de imagen. El nuevo tamaño establecido para las imágenes es de 800x800 ya que, es lo suficiente grande para disponer de suficientes datos con los que trabajar y aún así, funcionará una vez implementado en la Raspberry pi.

A continuación, se tratará de disminuir los problemas de ruido presente en la imagen mediante un filtro de tipo gaussiano, como ya se explicó en el apartado 7.4.

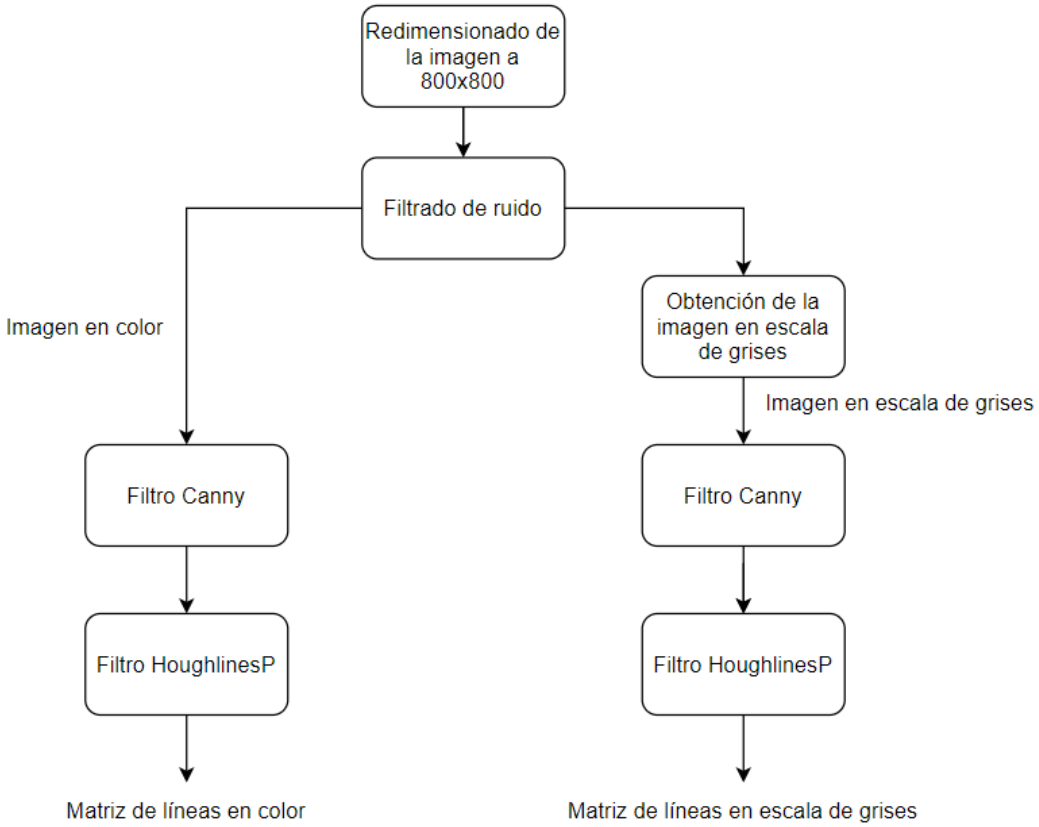


Figura 8.10 – Primeros pasos de procesado de la imagen

El filtro gaussiano escogido, es el "*Gaussianblur*" de OpenCV.

$$cv2.GaussianBlur(img, (n, n), sigma)$$

Cuando se toma la medida de realizar un filtrado u otro es muy importante destacar que esto puede variar según la cámara de procedencia de las imágenes. *Gaussianblur* demuestra ser eficiente tanto con las imágenes simuladas como con las fotografías de buena calidad obtenidas por un teléfono móvil pero, con las imágenes extraídas del repositorio de Pixabay será necesario añadir un filtro a mayores.

El filtro gaussiano es, además de un filtro de suavizado, un filtro lineal. Este filtro produce un suavizado más uniforme que otros como el de la media y provoca una ligera pérdida de detalles junto con una pequeña pérdida de nitidez en la imagen. Los parámetros a ajustar serán el número de filas y columnas de la matriz en la que se ejecuta,n, y el valor de sigma que cuanto mayor sea más agresivo será el filtro.

Tal y como se muestra aplicado el filtro gaussiano en la Figura 8.12, se eliminan suficientes interferencias sin perder demasiada calidad en los bordes buscados. Esto puede ser comprobado en la Figura 8.44, pues en ella se han usado los mismos parámetros para el filtrado de ruido.

Figura 8.11 – Laboratorio 1 en escala de grises

Figura 8.12 – Laboratorio 1 con filtro *Gaussianblur*

El filtro gaussiano realiza una media ponderada de los píxeles en una matriz cuadrada de n filas y n columnas. Los valores más cercanos al centro de la matriz tendrán mayor importancia en la media mientras que los más cercanos a los bordes tendrán una importancia menor.

Este filtro no solo requiere de la realización de matrices cuadradas, también tendrán que tener un píxel central, por lo que el número de filas y columnas,n, deberá ser impar.

Sigma indicará cuánto se pueden desviar los valores respecto al centro de la campana de gauss. Es recomendable poner un cero en el valor de sigma, esto hará que sea el programa quien elija el valor más aconsejable según la matriz seleccionada.

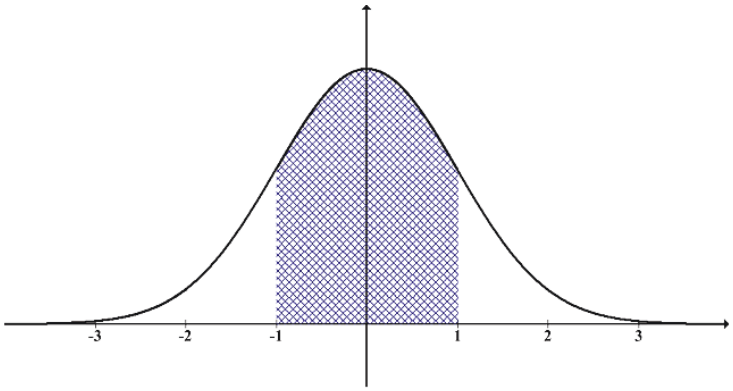


Figura 8.13 – Distribución normal

8.5. Filtro Canny

Este filtro es un detector de bordes. Al aplicarlo, se eliminará de la imagen toda información que resulte inútil para el proyecto y dibujará, en blanco sobre un fondo negro, todos los bordes presentes. Será necesario aplicar este filtro para que, una vez se busquen líneas rectas en la imagen, no se detecten infinidad de líneas no válidas.

$$cv2.Canny(imagen, hist1, hist2, apertureSize = 3)$$

Antes de la aplicación del filtro *Canny*, se realizará la conversión de la imagen a escala de grises y se les aplicará este filtro por separado a esta y a la imagen a color, ya que los resultados variarán.

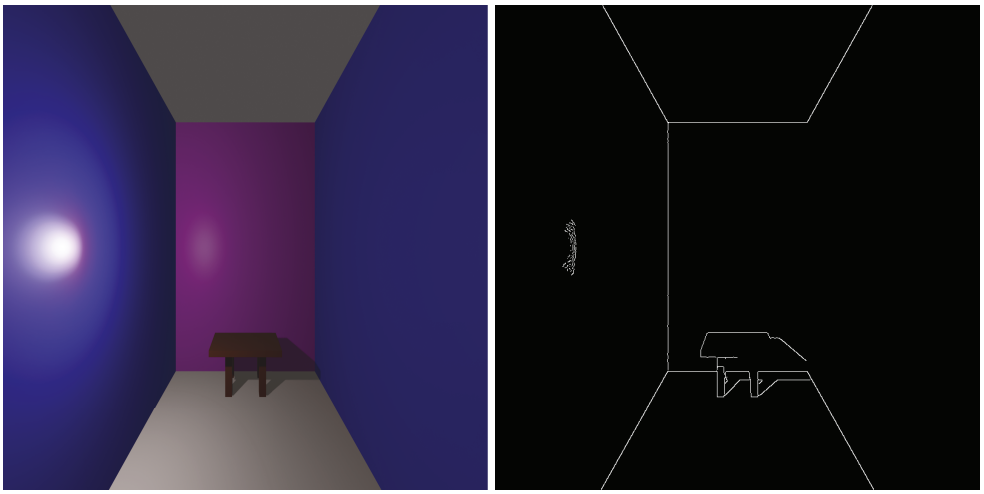


Figura 8.14 – Cuarto virtual con mesa, iluminación lateral y filtro *Canny* en escala de grises

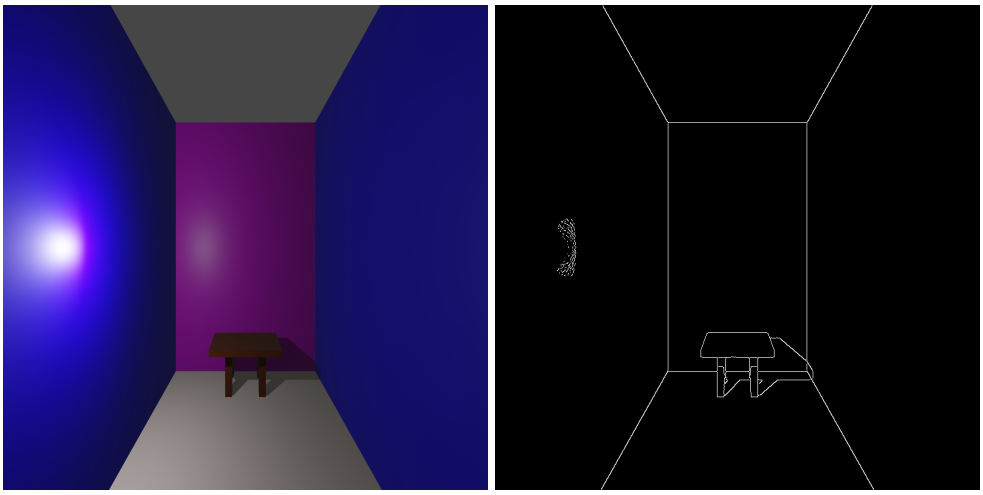


Figura 8.15 – Cuarto virtual con mesa, iluminación lateral y filtro *Canny* en color

El filtro *Canny* fue creado con el objetivo de cumplir tres propósitos que lo hacen ideal como base en este tipo de trabajo:

- Tener un alto porcentaje de acierto detectando solo bordes existentes.
- La distancia entre los píxeles de los bordes detectados y los reales debe ser mínima.
- Trata de detectar tan solo una respuesta por cada borde.

Este filtro realiza una serie de pasos internos que, aunque no se programen, es bueno conocer:

1. Realiza un filtrado gaussiano. Además del filtrado que se realiza en el tratamiento inicial de las imágenes [8.4], el propio filtro *Canny* incluye uno a mayores. Es necesario aplicar previamente un filtro gaussiano porque el incluido en el filtro *Canny* no se puede ajustar y resulta insuficiente para eliminar el ruido.
2. Produce un proceso análogo al que haría la detección de bordes de Sobel, sin entrar en detalle, este filtro busca cambios de intensidad mediante el cálculo de la primera derivada.
3. Se produce una supresión de no-máximos que consistirá en quedarse con aquellos límites cuyo grosor sea 1.
4. Por último, se aplica un umbral de histéresis. Este paso será el que resulte importante a la hora de configurar la función para el código.

Para asignar los valores al umbral de histéresis hay que saber cómo funciona. Se asigna un valor máximo y uno mínimo. Si el valor de un píxel es mayor que el valor máximo, este pertenecerá al límite, mientras que, si es inferior al valor mínimo, no lo hará. Por último, en caso de que un píxel tenga un valor situado entre el máximo y el mínimo, formará parte del límite siempre que esté en contacto con otro píxel perteneciente a él.

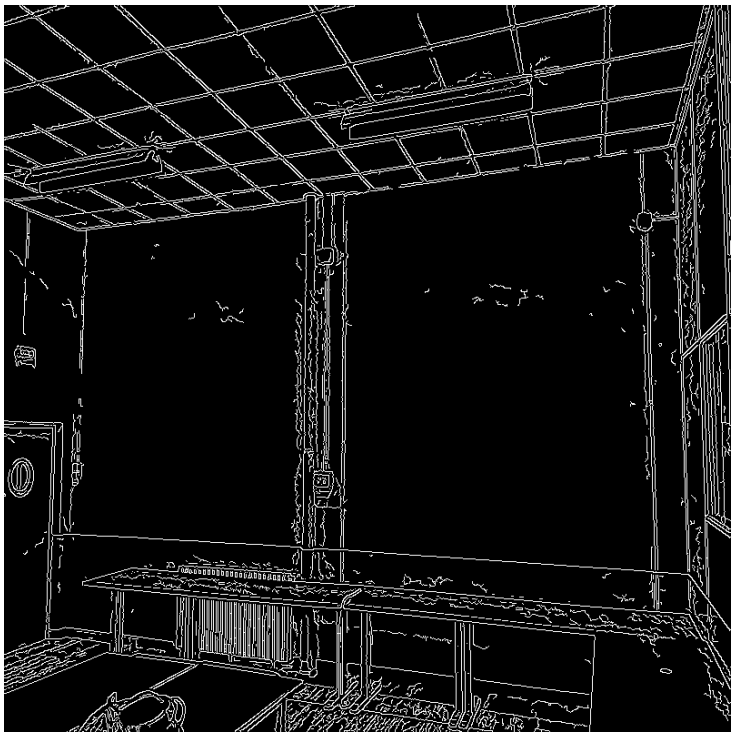


Figura 8.16 – Laboratorio 1 con muchos límites

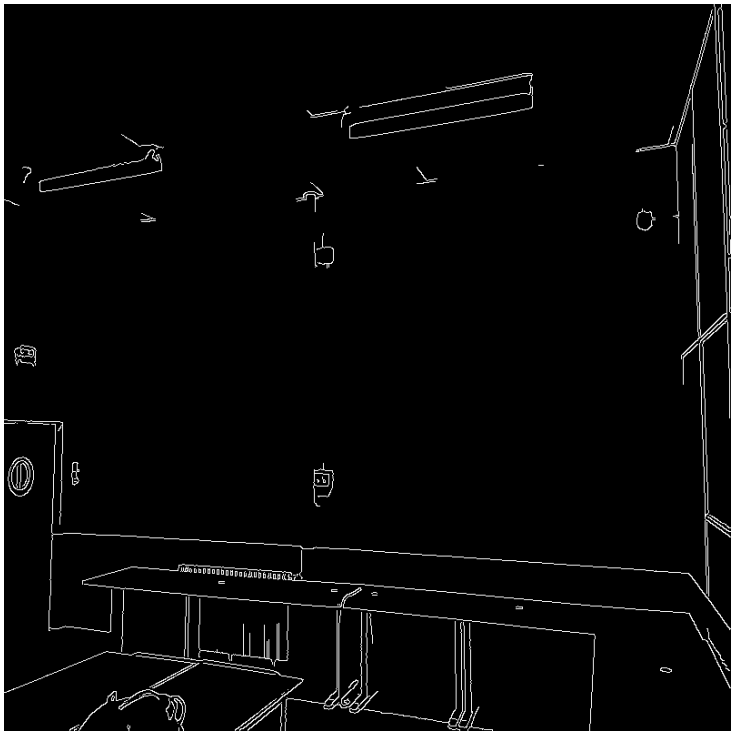


Figura 8.17 – Laboratorio 1 con pocos limites

Desde la Figura 8.18 hasta la Figura 8.25, se muestra a la izquierda el resultado del filtro *Canny* en escala de grises y a la derecha el resultado a color. Las Figuras 8.18, 8.19 y 8.20 se corresponden a las fotografías del laboratorio, mientras que, las Figuras 8.21, 8.22, 8.23, 8.24 y 8.25 serán las imágenes de prueba de Pixabay.

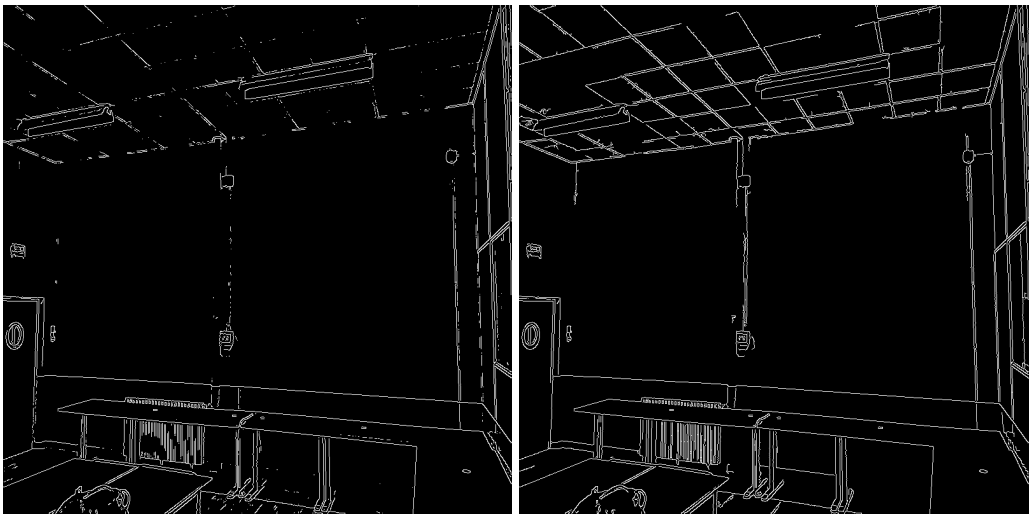


Figura 8.18 – Laboratorio 1 con filtro *Canny*

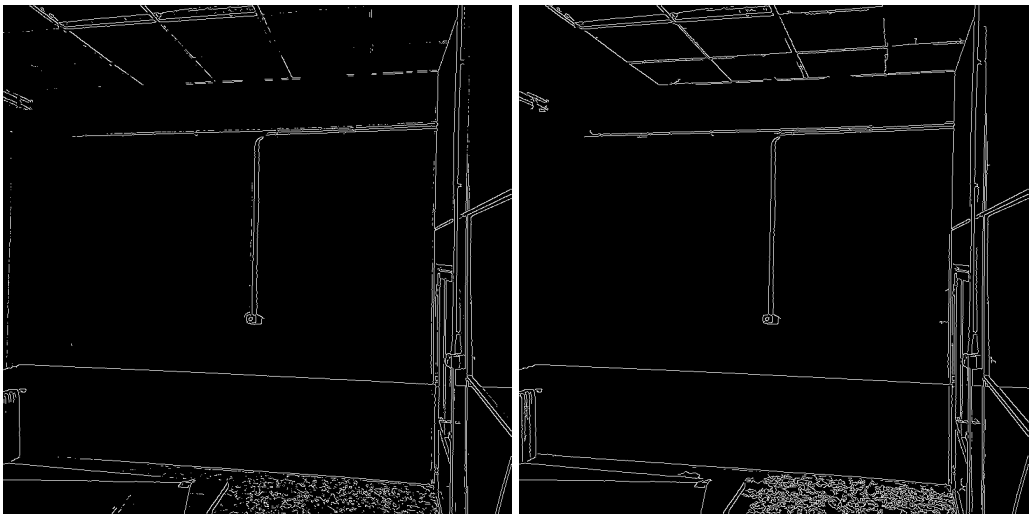


Figura 8.19 – Laboratorio 2 con filtro *Canny*

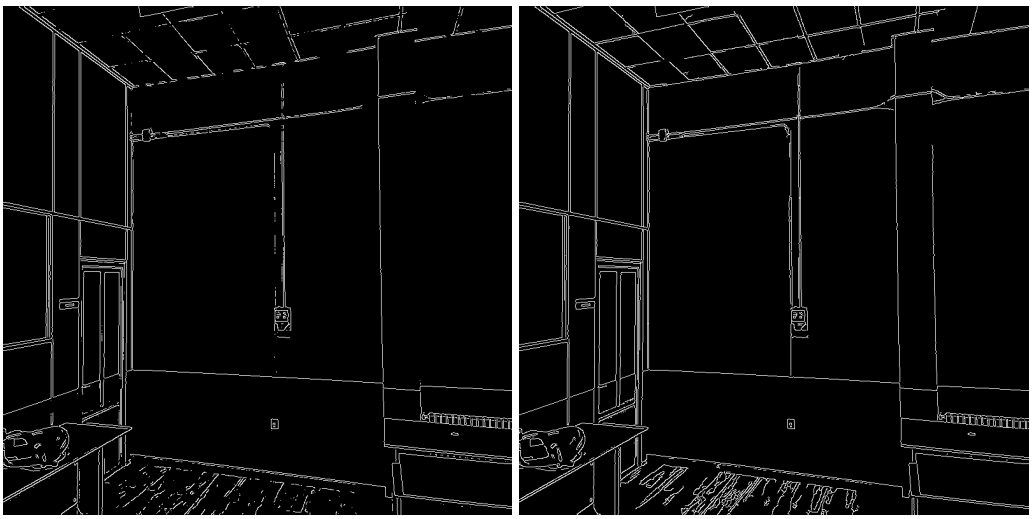


Figura 8.20 – Laboratorio 3 con filtro *Canny*

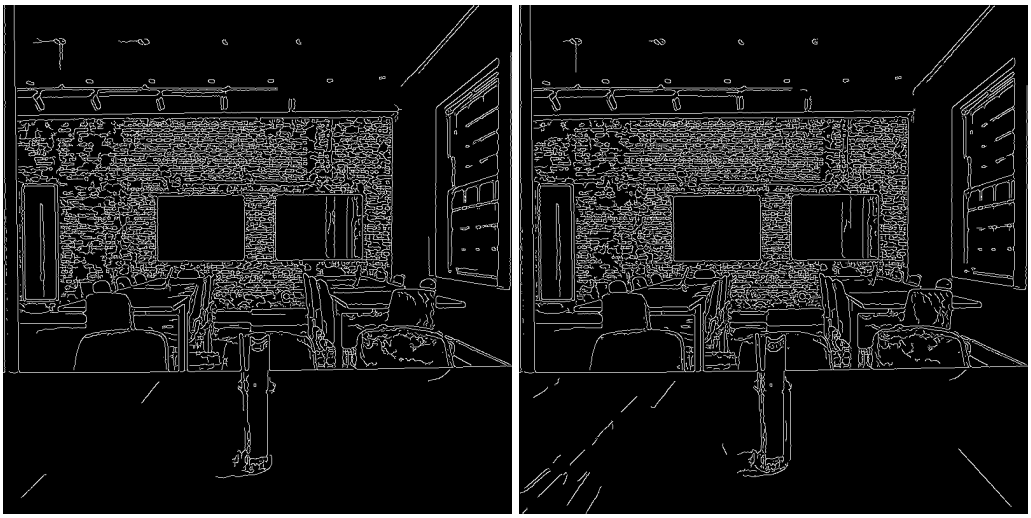


Figura 8.21 – Habitación 1 con filtro *Canny*

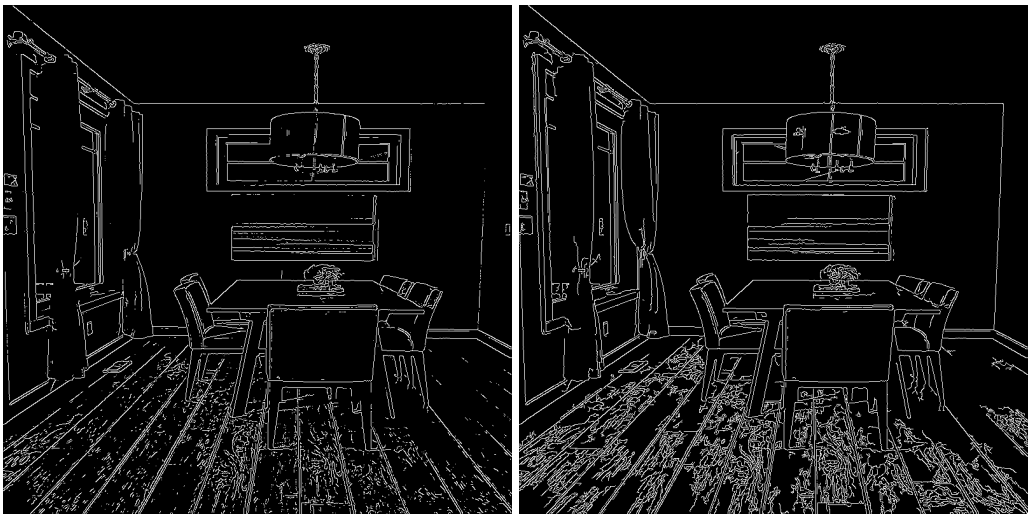


Figura 8.22 – Habitación 2 con filtro *Canny*

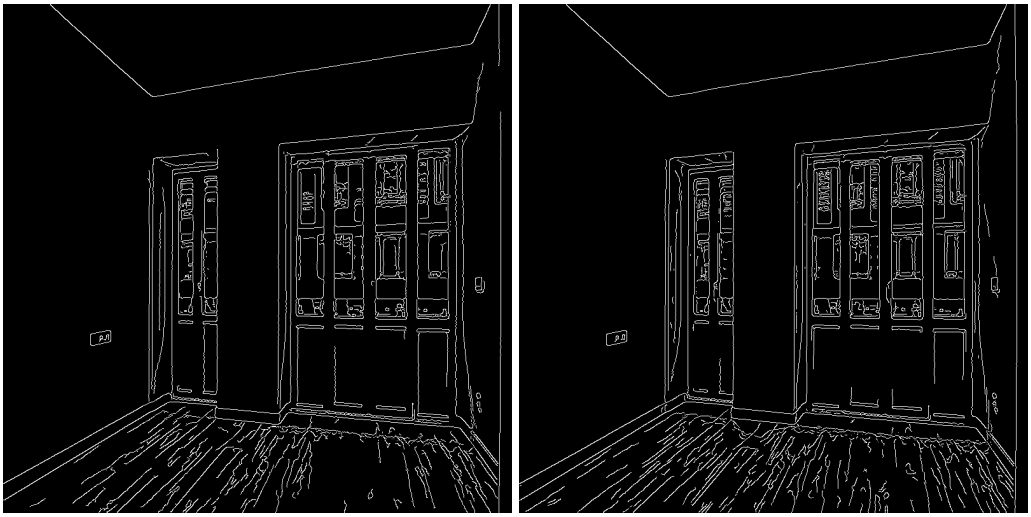


Figura 8.23 – Habitación 3 con filtro *Canny*

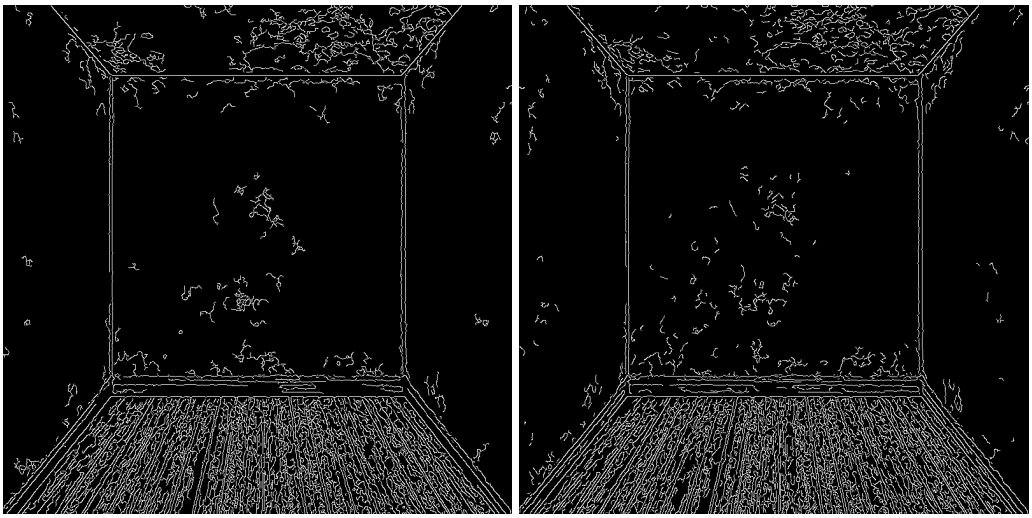


Figura 8.24 – Habitación 4 con filtro *Canny*

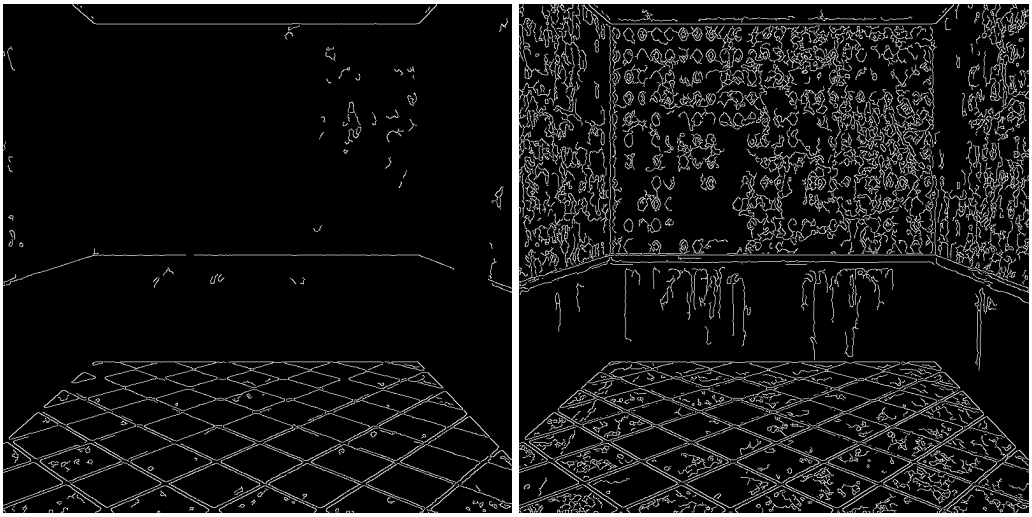


Figura 8.25 – Habitación 5 con filtro *Canny*

8.6. Filtro HoughlinesP

Una vez las imágenes están reducidas a los contornos de los objetos, se podrán detectar los límites con mayor sencillez, eliminando la mayor parte de problemas que pudieran surgir y reduciendo considerablemente el número de líneas halladas, siendo estas más precisas.

Esta función busca segmentos rectos y los devuelve en coordenadas cartesianas. Aplicándola sobre los bordes detectados previamente con el filtro *Canny*, se obtendrán líneas rectas que únicamente correspondan a bordes presentes en la imagen aunque, para que esto se cumpla, dependerá del ruido y de lo correctos que sean los parámetros con los que se configuren los filtros.

En las siguientes imágenes se puede observar un caso extremo de la diferencia entre aplicar este filtro sin aplicar *Canny* antes (Figura 8.26) y aplicándolo (Figura 8.27).

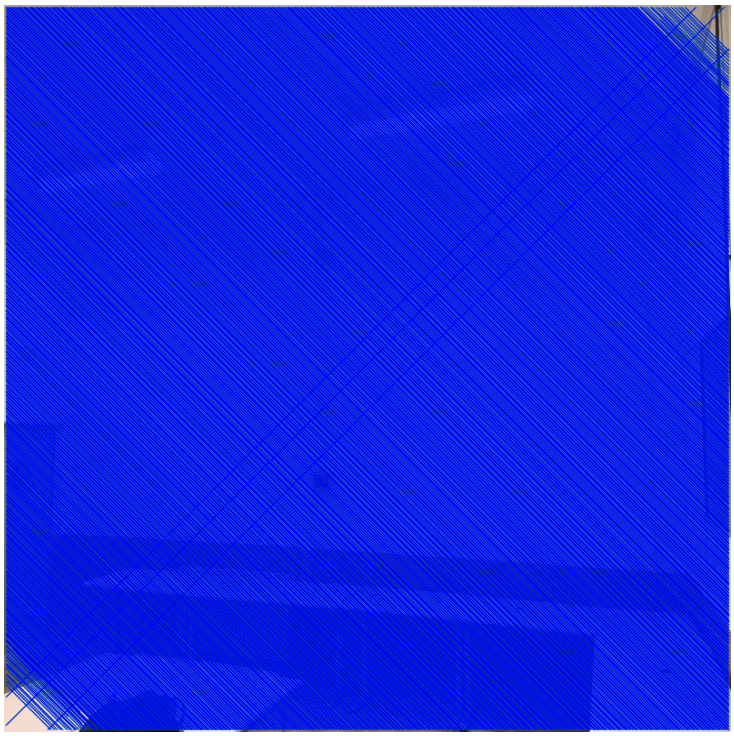


Figura 8.26 – Laboratorio 1 con filtro *HoughlinesP* sin *Canny*

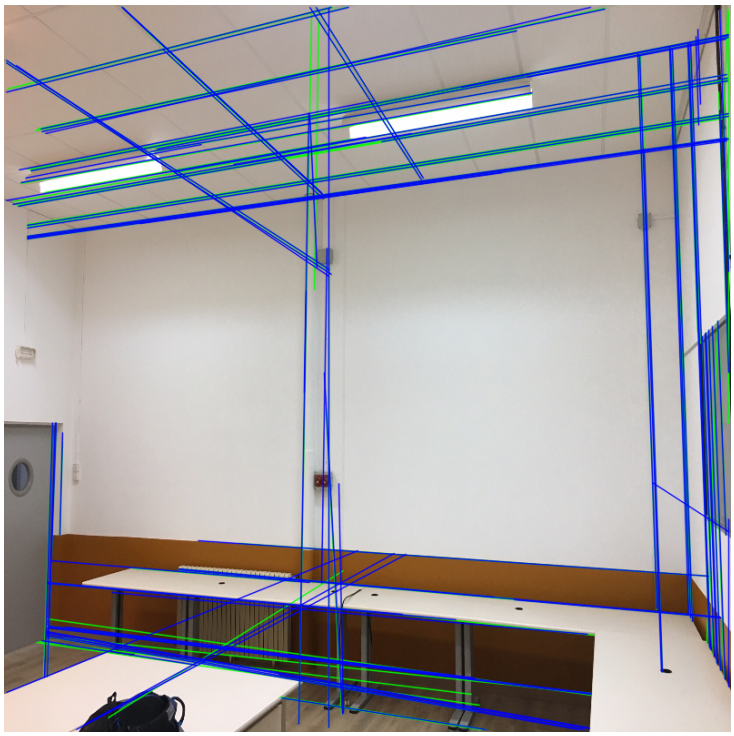


Figura 8.27 – Laboratorio 1 con filtro *HoughlinesP* con *Canny*

Con el filtro *Canny* aplicado antes y con parámetros de histéresis en 20 de mínimo y 50 de máximo, se aplica *HoughlinesP* a las otras dos imágenes del laboratorio.

Figura 8.28 – Laboratorio 2 con filtro *HoughlinesP* con *Canny*

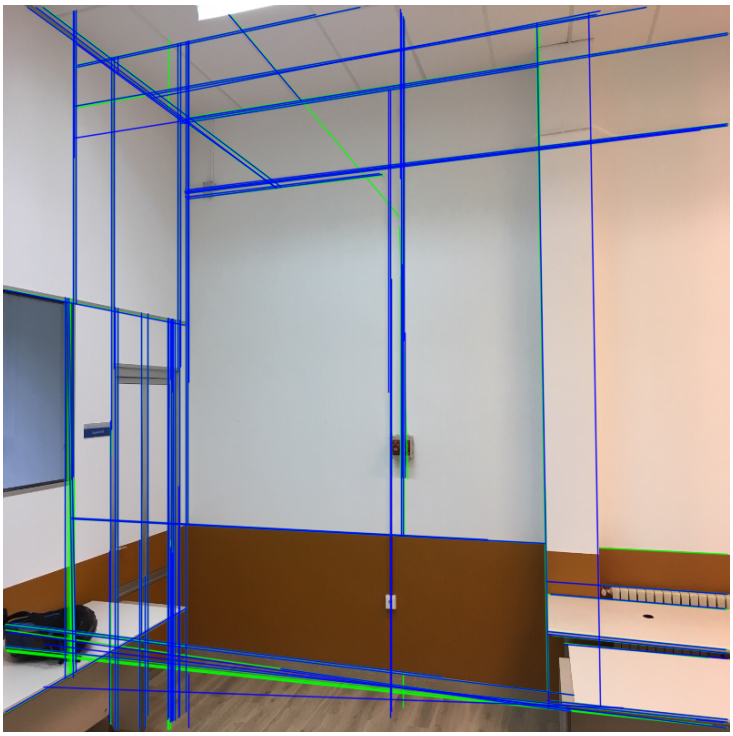


Figura 8.29 – Laboratorio 3 con filtro *HoughlinesP* con *Canny*

Se busca realizar todo el programa para unos valores fijos en los parámetros establecidos para configurar el filtro. Se elige para ello un valor elevado de máxima distancia permitida entre dos líneas y un valor aproximadamente de diez en el tamaño mínimo para que una línea sea tomada en cuenta.

$$cv2.HoughLinesP(edges1, 1, np.pi/180, 100, minLineLength = 10, maxLineGap = 10)$$

Los parámetros asignados en el filtro *Canny* muestran su importancia en el número de líneas que se encontrarán en este filtro. En las dos siguientes Figuras se puede observar la diferencia entre haber establecido unos valores en el umbral de *Canny* mayores o menores. En la Figura 8.30 se han colocado unos valores elevados para la histéresis del filtro *Canny*, lo que hace a la función *HoughlinesP* detectar menos líneas que en la Figura 8.31, para la que se habrá colocado un valor para la histéresis mucho menor.

Figura 8.30 – Laboratorio con filtro *HoughlinesP* con histéresis alta en *Canny*

Figura 8.31 – Laboratorio con filtro *HoughlinesP* con histéresis baja en *Canny*

Al principio, la función se prueba con valores mayores en el tamaño mínimo de las líneas. Esto provoca el problema de que, además de detectar las líneas correctas, se generan líneas por sombras pequeñas. Al haber más líneas con la dirección de la luz y no del suelo, se dificulta mucho para el programa la búsqueda de la pendiente correcta.

Tras aplicar este filtro a ambas imágenes, a color y en escala de grises, se dispondrá de dos matrices con los valores iniciales y finales en cada línea a modo de coordenadas cartesianas.

8.7. Función trazalineas

Para la búsqueda de las líneas más importantes que delimiten las superficies, se crea una función que sea válida para múltiples casos con solo variar sus parámetros de entrada.

Esta función buscará localizar las siguientes líneas:

- Horizontal: Suelo-pared central.
- Horizontal: Pared central-techo.
- Vertical: Pared izquierda-pared central.
- Vertical: Pared central-pared derecha.
- Cuatro diagonales que salen de las uniones entre el resto de líneas. anteriores.

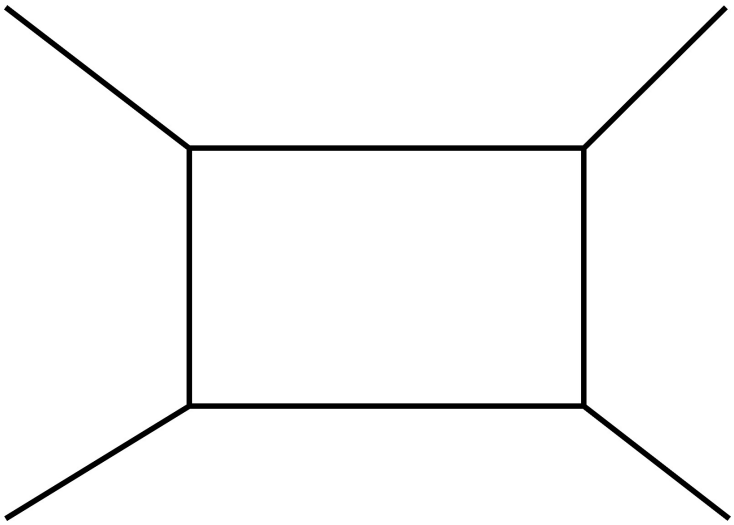


Figura 8.32 – Límites buscados

Todas estas líneas se tratan de localizar en la imagen a color y en escala de grises, por lo que se tienen dos resultados disponibles.

$$trazalineas(matriz, limiteyinf, limiteysup, limitexinf, limitexsup, pendientemin1, pendientemax1, pendientemin2, pendientemax2, orientacion, color)$$

Esta función tiene como parámetros de entrada:

- Rango de pendientes en el que se mueve la línea buscada: Pendientemin1, pendientemax1, pendientemin2 y pendientemax2 serán las entradas encargadas de fijar la pendiente buscada.
- Zona de la imagen: Es acotada por las entradas limiteyinf,limiteysup,limitexinf y limitexsup.
- Orientación: Se indica si la línea buscada es una vertical u horizontal, pues el código cambiaría para evitar errores.
- Color: Indica si se utiliza la imagen en escala de grises o a color.

Al crear una función que dependa de estos parámetros, se está permitiendo que el código pueda ser reutilizado para otras aplicaciones. Por ejemplo, con tan solo ajustar las pendientes, esta misma función podría utilizarse para buscar de forma muy efectiva líneas en una carretera.

Lo primero que hará la función será separar la imagen en zonas teniendo en cuenta las líneas que se van a buscar (Figura 8.33). La línea entre suelo y pared deberá estar por debajo de la mitad de la altura de la imagen, en otro caso la cámara estaría mal posicionada o la pared frontal demasiado alejada como para que el programa trabajase de forma adecuada.

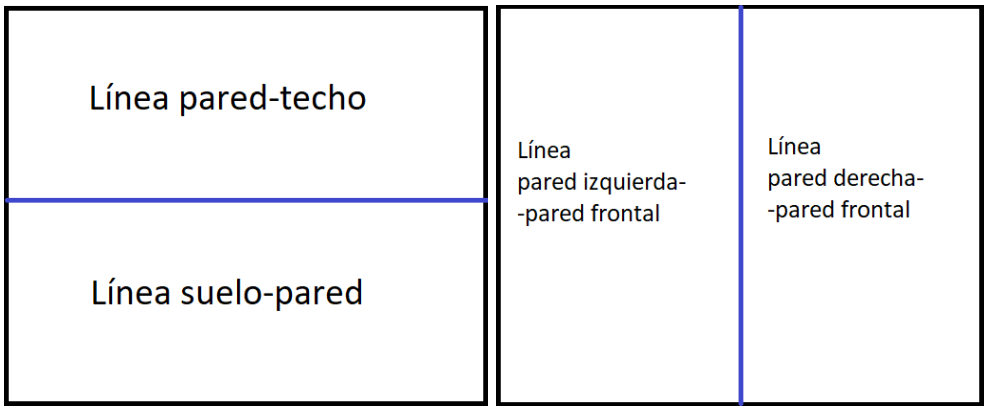


Figura 8.33 – División de la imagen en mitades para búsqueda de líneas.

Se dividirá así la imagen en mitades para las líneas verticales y horizontales, y en cuadrantes (Figura 8.34) para la búsqueda de cada diagonal. También se establecen unos límites razonables para la inclinación buscada en la pendiente de cada línea.

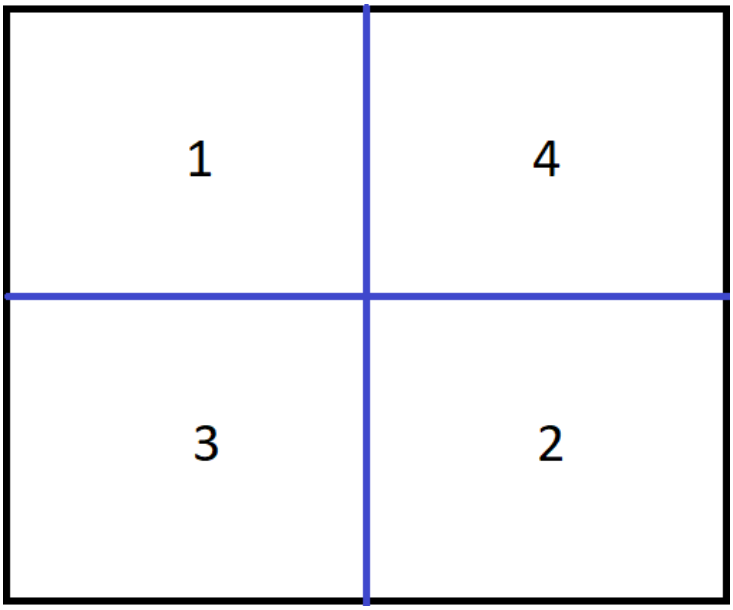


Figura 8.34 – División de la imagen en cuadrantes

La función ejecutará los siguientes pasos:

Figura 8.35 – Pasos de la función *trazalineas*

Si se ejecuta el código en el cuarto virtual, los resultados se mantendrán correctos excepto una vez introducida la mesa como parte del mobiliario. En la Figura 8.37 se puede observar como el programa trazaría líneas tanto para la mesa, la sombra y el límite buscado.

Aunque pueda parecer irrelevante, si se ejecutase el código en una habitación real se volvería imposible quedarse con la línea correcta. En el caso de esta imagen, ya se ve que habrá problemas con las diagonales, pues incluso tratándose de un cuarto virtual, la diagonal inferior derecha no se está dibujando completamente.

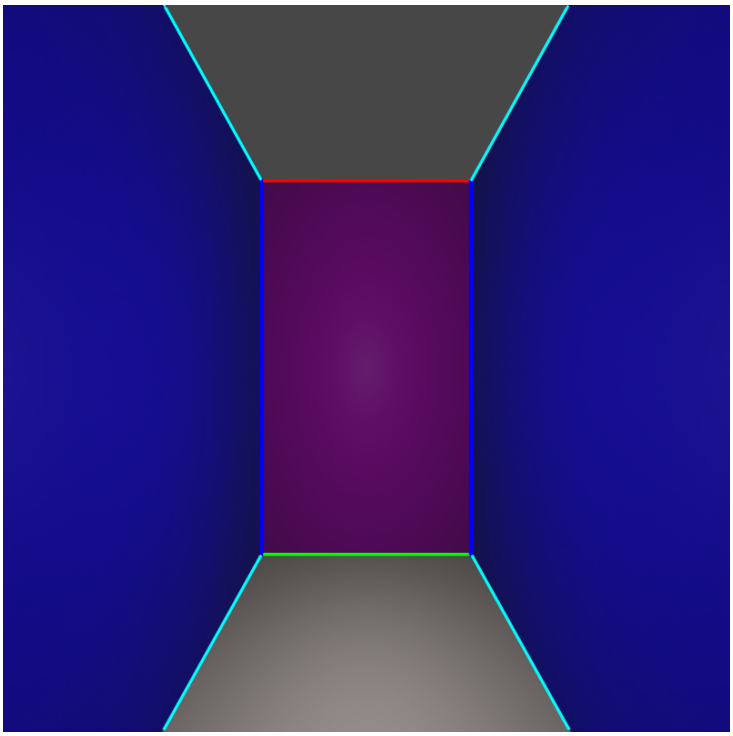


Figura 8.36 – Detección en cuarto virtual precisa

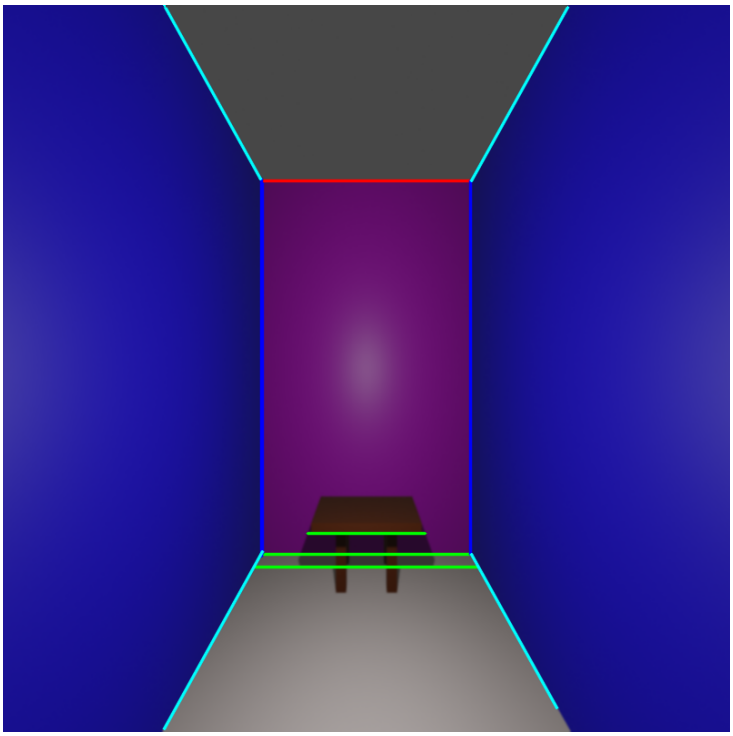


Figura 8.37 – Error por múltiples líneas detectadas en el suelo

Para solucionar este problema, habrá que comenzar a reducir las líneas obtenidas hasta obtener solamente límites válidos. Para esto, se realizan los pasos de la Figura 8.35 y, como se puede observar, la función se basará en los siguientes dos valores:

- Pendiente.

- Proyección de la línea en el margen.

Primero se calculará la pendiente de cada línea y, debido a la perspectiva, la mayoría de objetos, como pueden ser una mesa o una cama, coincidirán en pendiente con la línea que se esté buscando en cada momento.

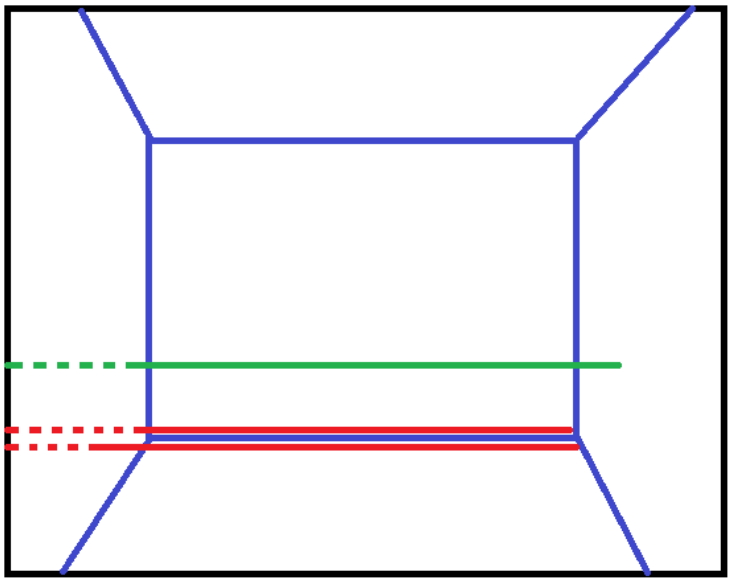


Figura 8.38 – Ejemplos de líneas proyectadas al margen de una imagen

La función se queda con las líneas que tienen la pendiente más repetida y calcula la proyección de cada una hasta el borde de la imagen. El programa se queda con las líneas que, con valores aproximados, tengan misma pendiente y misma proyección. En el caso de la Figura 8.38, se muestra como se realizaría la proyección al margen de las tres líneas detectadas (las rojas y la verde) y, el programa, guardaría en una matriz las dos rojas debido a la cercanía entre las proyecciones, desechando así la verde. Si bien esto no siempre será efectivo, es el método pensado más eficiente con las limitaciones de hardware.

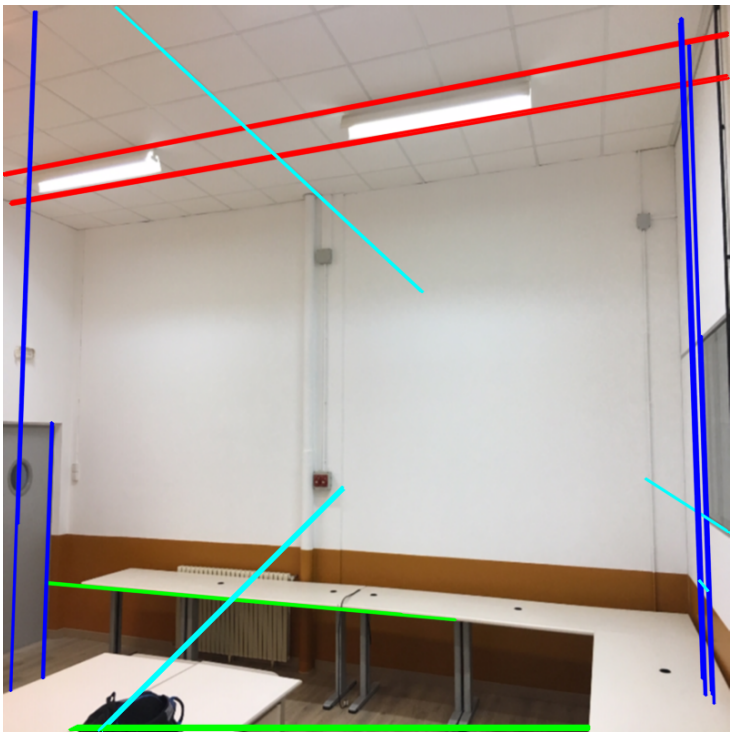


Figura 8.39 – Laboratorio 1 con límites filtrados

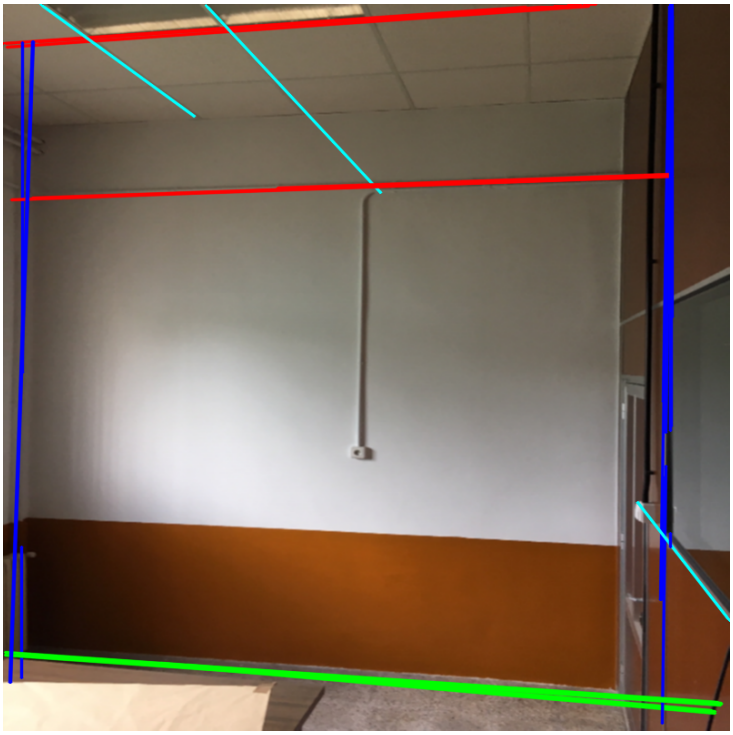


Figura 8.40 – Laboratorio 2 con límites filtrados

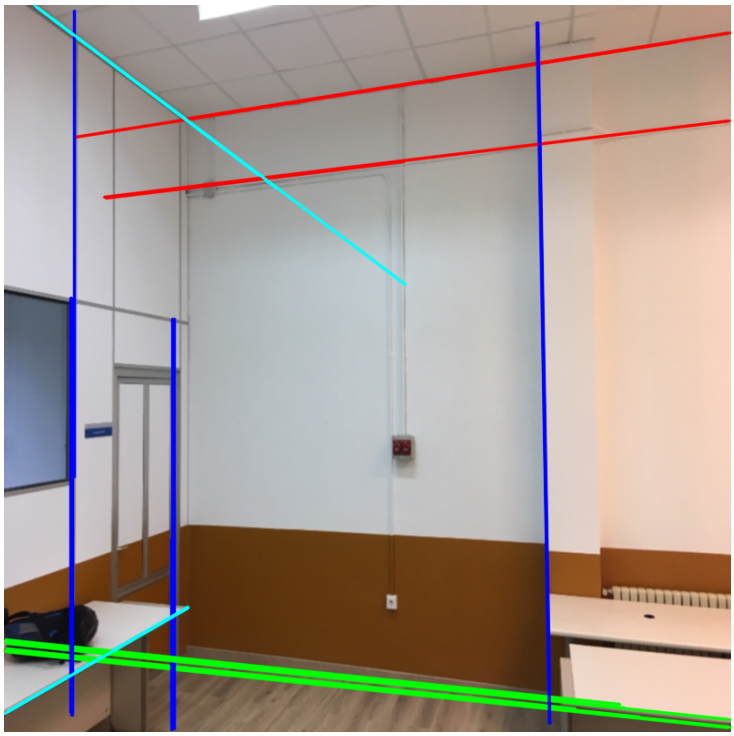


Figura 8.41 – Laboratorio 3 con límites filtrados

8.8. Punto de fuga y redimensionado

En muchas ocasiones resulta imposible encontrar todas las diagonales de la imagen solo con la función *trazalineas*. Para poder encontrarlas todas, habrá que deducirlas a partir de las existentes hallando el punto de fuga. En la imagen se muestra un ejemplo de dónde se situaría el punto de fuga y cómo se trazaría a partir de las diagonales de una habitación.

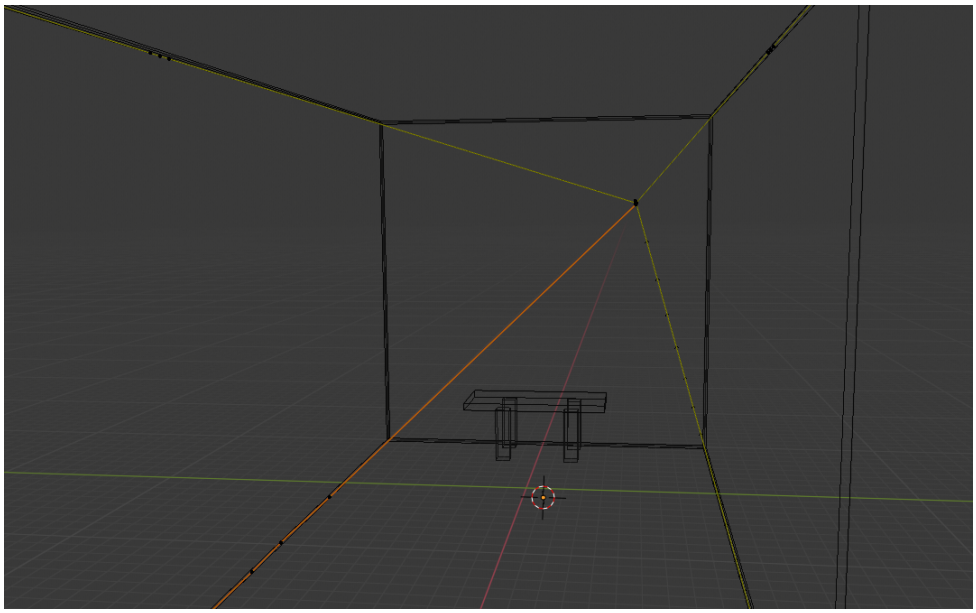


Figura 8.42 – Punto de fuga

Al haber trabajado con redondeos, si las diagonales se encontrasen en esquinas opuestas podrían no cruzarse o hacerlo demasiado lejos del punto de fuga real, por lo que el punto de fuga calculado se situará en un pequeño círculo de radio 5 píxeles.

Para realizar la búsqueda del punto de fuga, se usarán las diagonales calculadas tanto en la imagen a color como en la imagen en escala de grises. Las dos diagonales empleadas para obtenerlo deben cumplir dos condiciones:

- No pueden ser diagonales opuestas.
- No pueden estar en el mismo cuadrante (Figura 8.34).

Al igual que a la hora de escoger las líneas válidas, el punto de fuga también se buscará que sea el más repetido de entre todas las combinaciones de diagonales usadas para calcularlo.

Seleccionando la primera línea de cada tipo calculado (exceptuando las diagonales), se realizan los puntos de corte entre ellas para redimensionarlas. Desde esos puntos de corte y usando el punto de fuga, se podrán reconstruir todas las líneas diagonales de la imagen. Esta operación se repetirá dos veces, una para imagen a color y otra para escala de grises, entregando dos soluciones en función del tratamiento inicial de la imagen.

En amarillo se visualiza el resultado tras emplear la imagen a color y en negro el resultado de la imagen en escala de grises. En el caso de la Figura 8.51, el resultado de la imagen en escala de grises se muestra en azul debido a la dificultad para observar la línea suelo-pared en negro.

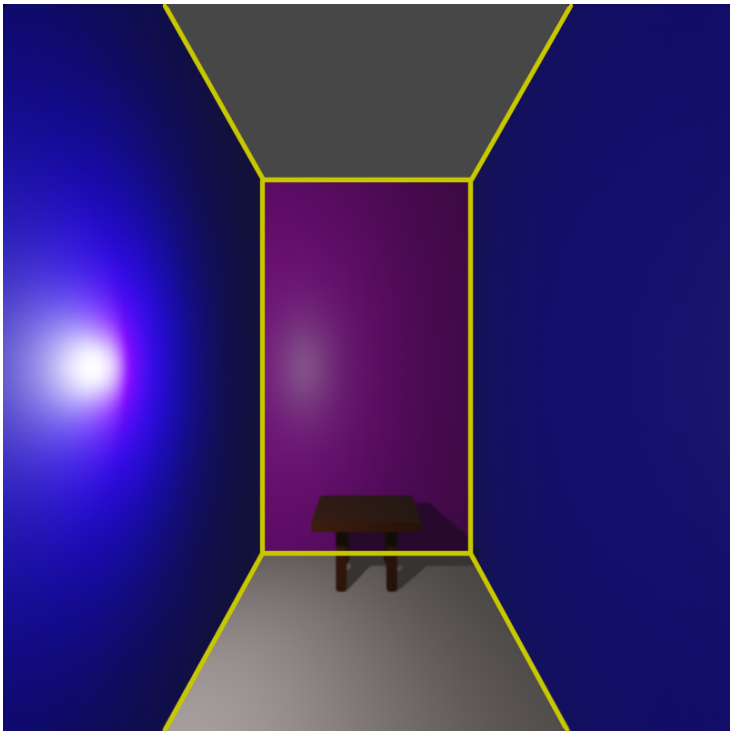


Figura 8.43 – Cuarto con mesa e iluminación lateral delimitado

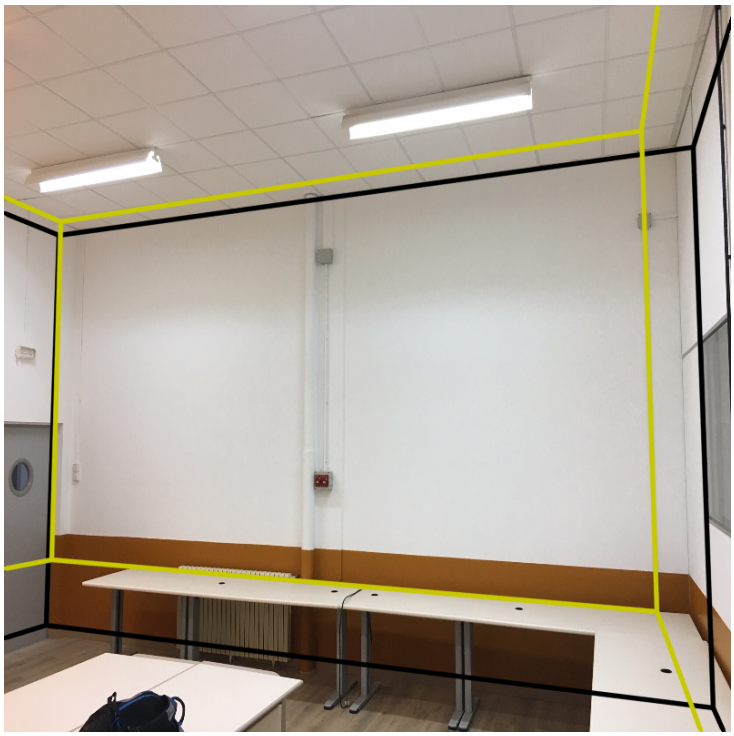


Figura 8.44 – Laboratorio 1 delimitado

Figura 8.45 – Laboratorio 2 delimitado

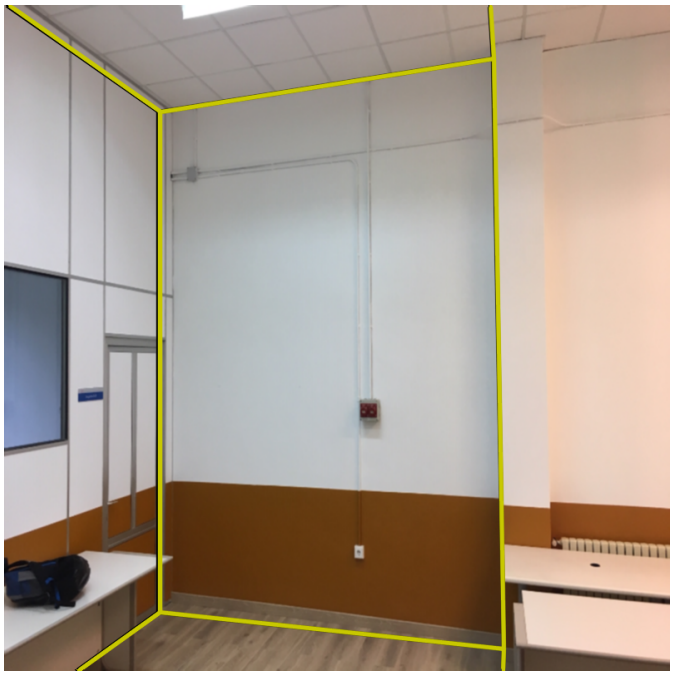


Figura 8.46 – Laboratorio 3 delimitado

Las siguientes imágenes han sido obtenidas del repositorio de Pixabay. En el mundo de la fotografía, es muy común añadir filtros para hacerlas parecer más artísticas, el problema es que este tipo de filtros añaden ruido a mayores, como puede ser el de sal y pimienta y el uniforme. Según la cantidad de ruido que hayan añadido llega a resultar un gran problema, por lo que para estas fotos, se aplicará el filtro de ruido *Medianblur* a mayores del gaussiano. Es un filtro de mediana [7.4].

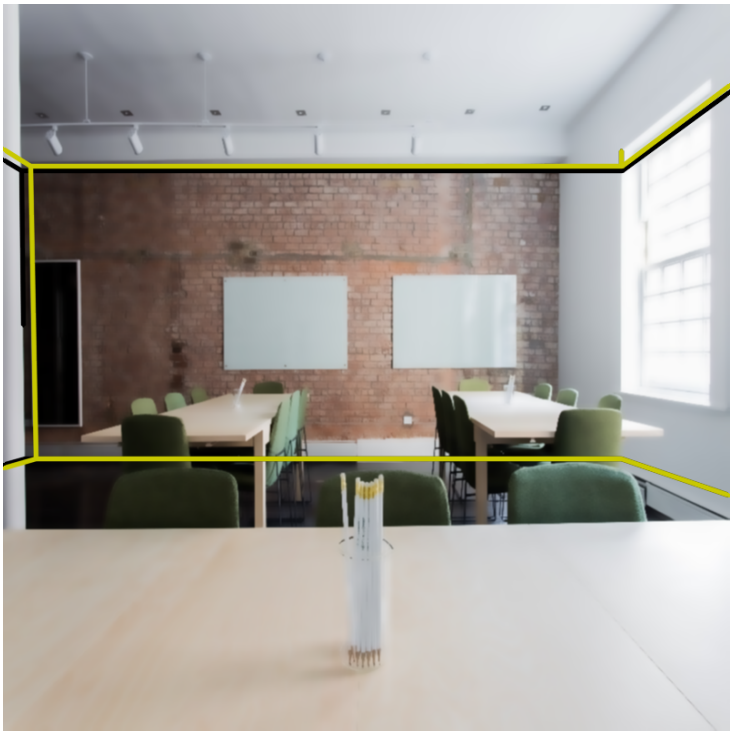


Figura 8.47 – Prueba 1 de habitación delimitada

Figura 8.48 – Prueba 2 de habitación limitada

Figura 8.49 – Prueba 3 de habitación delimitada

Figura 8.50 – Prueba 4 de habitación delimitada

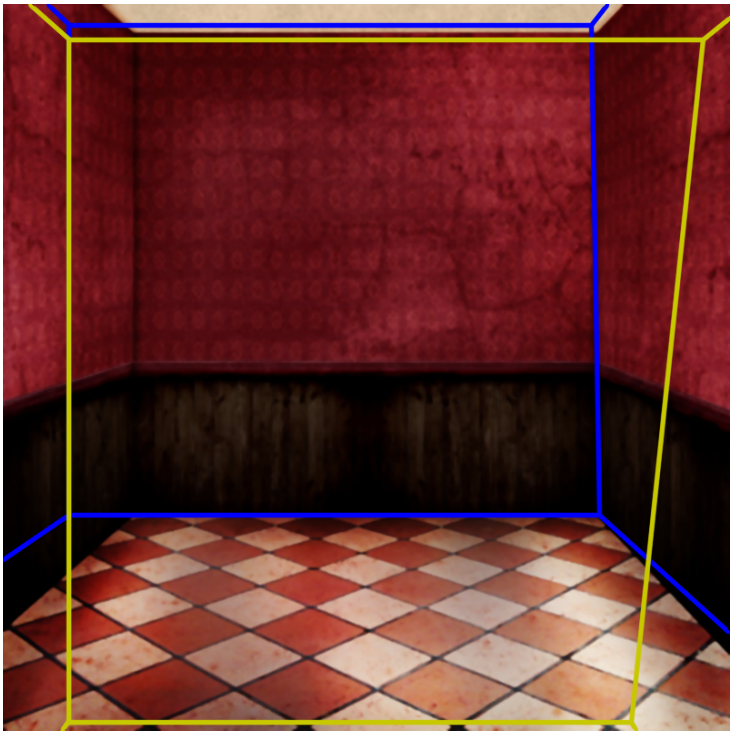


Figura 8.51 – Prueba 5 de habitación delimitada

8.9. Superposición de una imagen

Una de las funciones que se podrían lograr en el caso de identificar correctamente los límites entre paredes, suelo y techo, sería el cambio de textura de una de las superficies.

Se seleccionan diferentes superficies del repositorio de Pixabay para utilizarlas como nueva textura. Dos de las nuevas texturas que se emplearán se muestran como ejemplo en la siguiente Figura:

Figura 8.52 – Pared de ladrillos y pared de madera azul

Para lograr esta aplicación, una vez se encuentran la superficies delimitadas, se creará una máscara de la nueva textura de manera que encaje con los límites hallado cambiando a negro el valor de los píxeles que quedan en el exterior de los bordes. El proceso puede verse en las siguientes Figuras:

Figura 8.53 – Paso 1 y 2 de la creación de la máscara

Figura 8.54 – Paso 3 y 4 de la creación de la máscara

En la Figura 8.53, se vuelven negro primero los píxeles que están por encima de la ecuación de la recta del límite pared-techo y después los píxeles que se sitúan por debajo de la del límite suelo-pared. En la Figura 8.54 se realiza el mismo proceso con las líneas verticales de la izquierda y de la derecha.

Se crea una máscara con el resultado del corte y se emplea, junto con la imagen de la textura original y la habitación delimitada, para dar lugar al resultado deseado. Una vez visualizados los límites a color y en escala de grises, se solicita al usuario del programa la selección de los límites en los que se situará la pared.

Estos son algunos resultados:

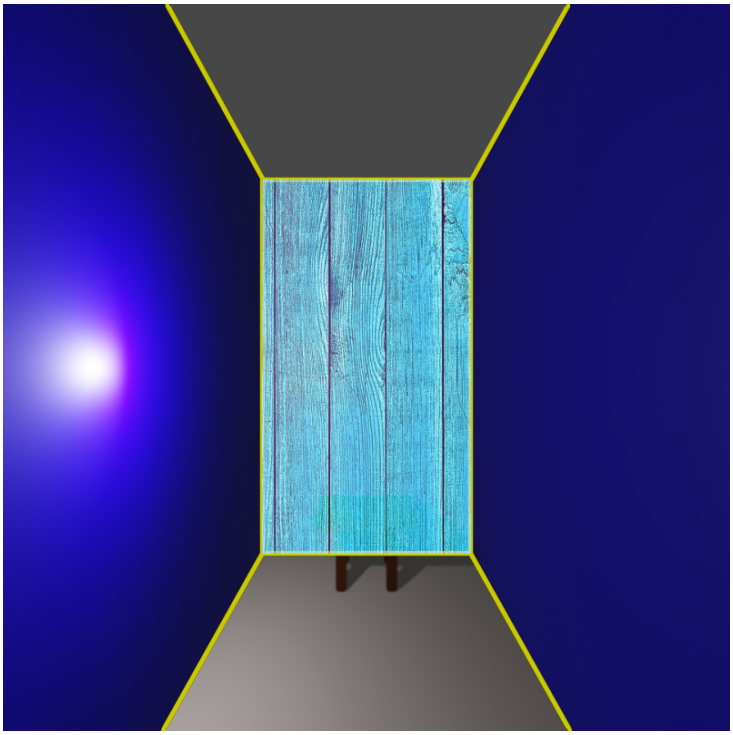


Figura 8.55 – Cuarto virtual con mesa y luz lateral con cambio de pared

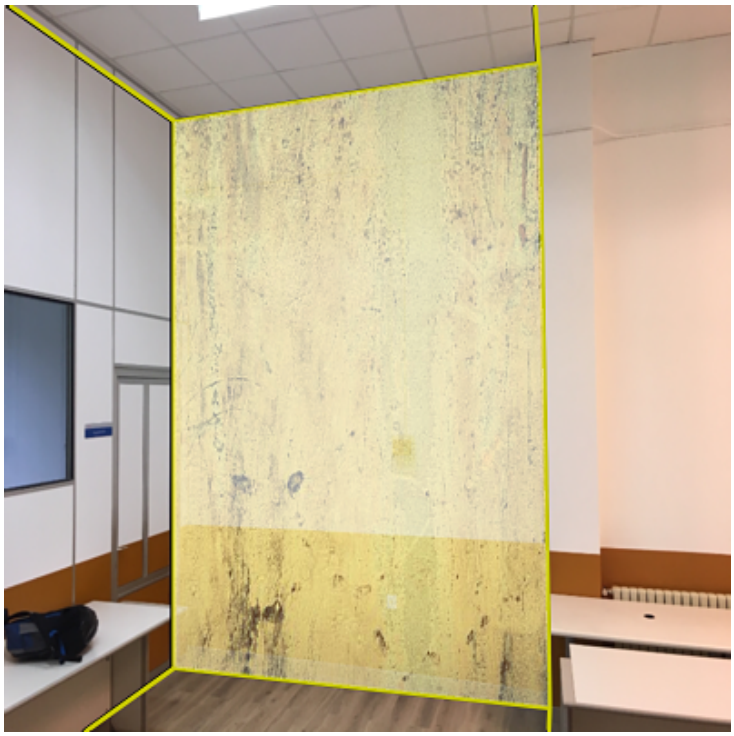


Figura 8.56 – Laboratorio 1 con cambio de pared

Figura 8.57 – Laboratorio 3 con cambio de pared

Figura 8.58 – Prueba 1 con cambio de pared

Figura 8.59 – Prueba 2 con cambio de pared

Figura 8.60 – Prueba 3 con cambio de pared

Figura 8.61 – Prueba 4 con cambio de pared

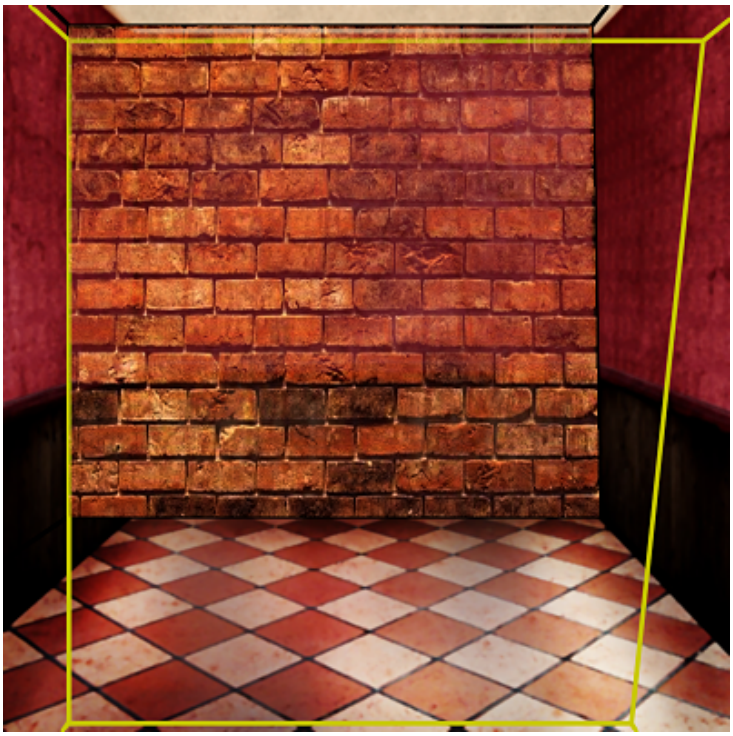


Figura 8.62 – Prueba 5 con cambio de pared

En la Figura 8.63, se muestra de forma general las distintas fases por las que pasará una imagen:

1. Imagen inicial que será procesada.

2. Aplicación del filtro *Canny* tras el tratamiento inicial de la imagen.

3. Detección de líneas rectas con el filtro *HoughlinesP*.

4. Aplicación de la función trazalineas para quedarse con los límites detectados más importantes.

5. Delimitación de las superficies.

6. Cambio de la superficie presente en la pared frontal mediante la aplicación de una máscara.

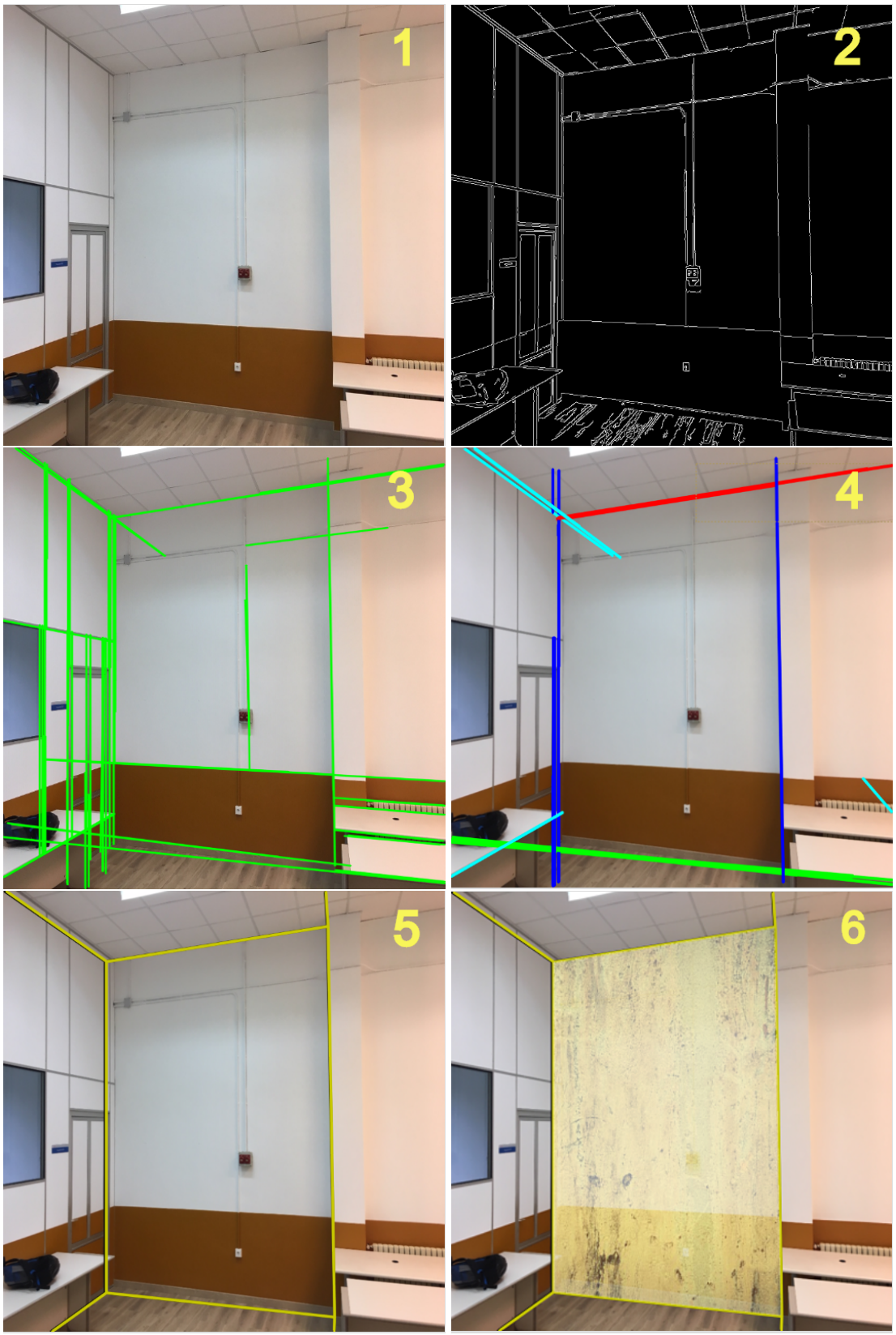


Figura 8.63 – Proceso completo aplicado sobre el laboratorio 3

8.10. Aplicación de realidad aumentada

Conectando una cámara a la Raspberry e incorporando el programa, se podrá monitorizar por medio del programa VNC como una aplicación de realidad aumentada desde el ordenador portátil. Para implementar el sistema de forma completamente independiente se podrá conectar a la Raspberry un monitor, un ratón y un teclado. De esta forma no será necesario un ordenador para ejecutar el programa y visualizarlo.

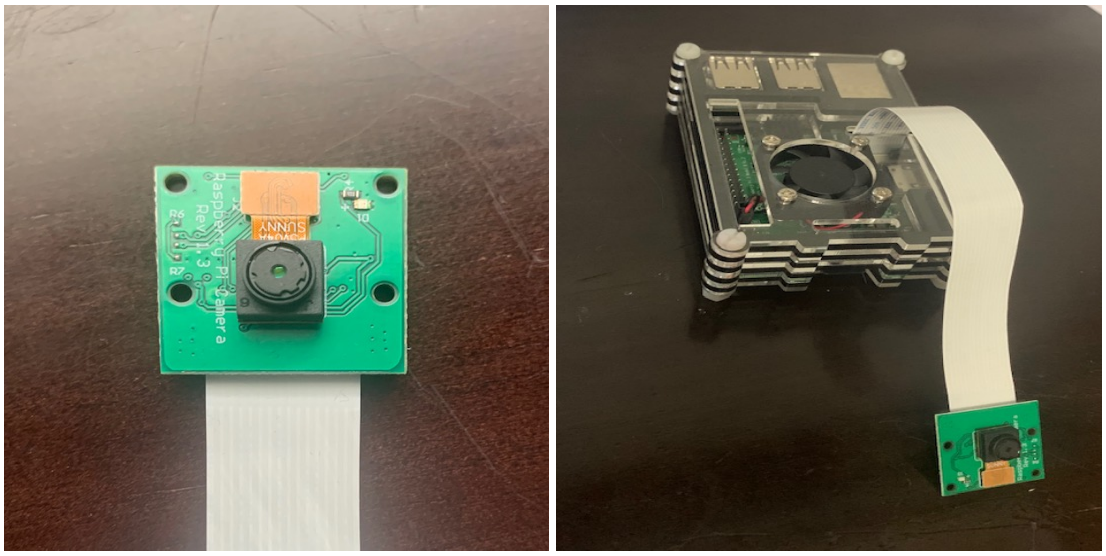


Figura 8.64 – Cámara y montaje en Raspberry Pi 3 modelo B

Con el código ejecutándose de forma óptima en las imágenes disponibles, se traslada la prueba a la Raspberry Pi. Se configurará que la cámara tome fotografías cada vez que se inicie el código y las guarde siempre con un mismo nombre, así el programa va tomando la nueva imagen, ejecutando el código sobre ella y mostrándolo en bucle hasta una vez cortada su ejecución.

Al tratarse nuevamente de una cámara diferente a la utilizada en un inicio en la prueba del TFG, se prueba el sistema únicamente con el filtro *Gaussianblur* para evitar el ruido. El motivo de dejar de usar el filtro *Medianblur* es que, en la práctica, cuando se emplea con una cámara de peor calidad, se difumina demasiado la imagen y el sistema se vuelve incapaz de detectar bordes correctamente.

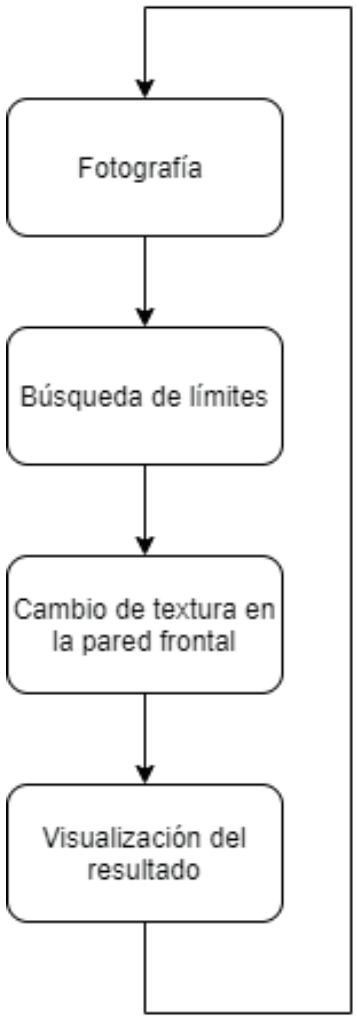


Figura 8.65 – Funcionamiento sistema de realidad aumentada

La aplicación es configurada para que siempre que sea incapaz de delimitar una habitación por ausencia de diagonales suficientes, muestre las líneas de las que disponía hasta el momento. Si no dispone de líneas que mostrar, el programa seguirá tomando fotografías hasta hallar una que considere válida.

Para realizar una prueba en la vida real se han colocado libros a modo de paredes y techo sobre una mesa. Así, se pude probar el dispositivo en un entorno controlado a pesar de las limitaciones existentes.

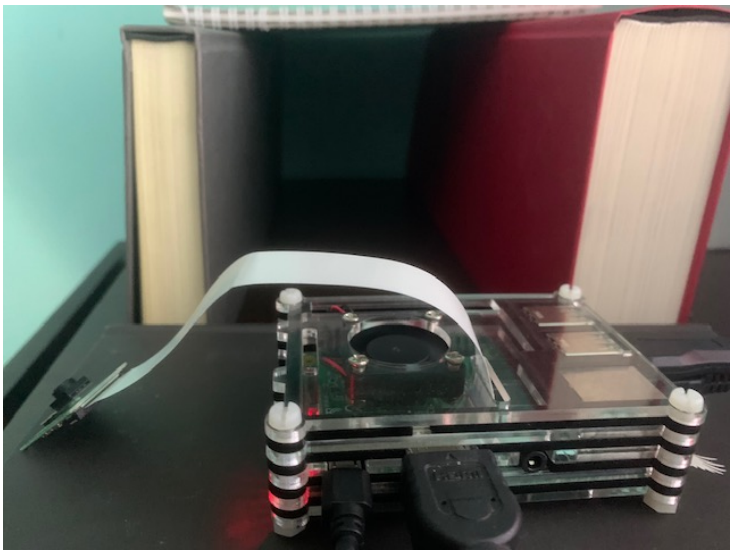


Figura 8.66 – Montaje de prueba del sistema

A continuación se muestran algunos resultados variados obtenidos por la aplicación:

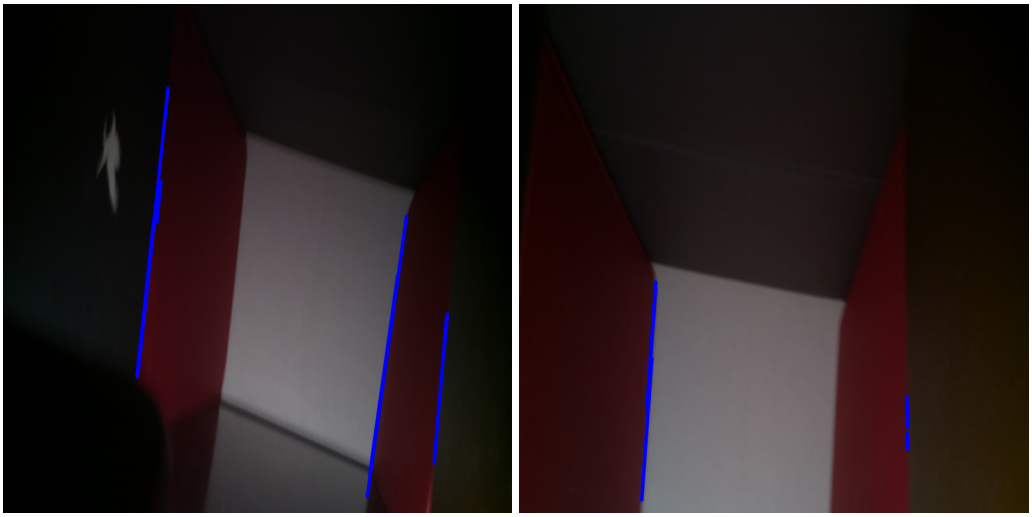


Figura 8.67 – Error forzado por iluminación y perspectiva

En las dos imágenes de la Figura 8.67, se produce un error debido a la iluminación y a la perspectiva desde la que se toma la fotografía. Se deduce que, debido a la falta de luz, no es capaz de distinguir ninguna diagonal y, por disponer de una inclinación muy grande (además del fallo de iluminación), las líneas de suelo y techo no entran dentro de los parámetros de búsqueda.

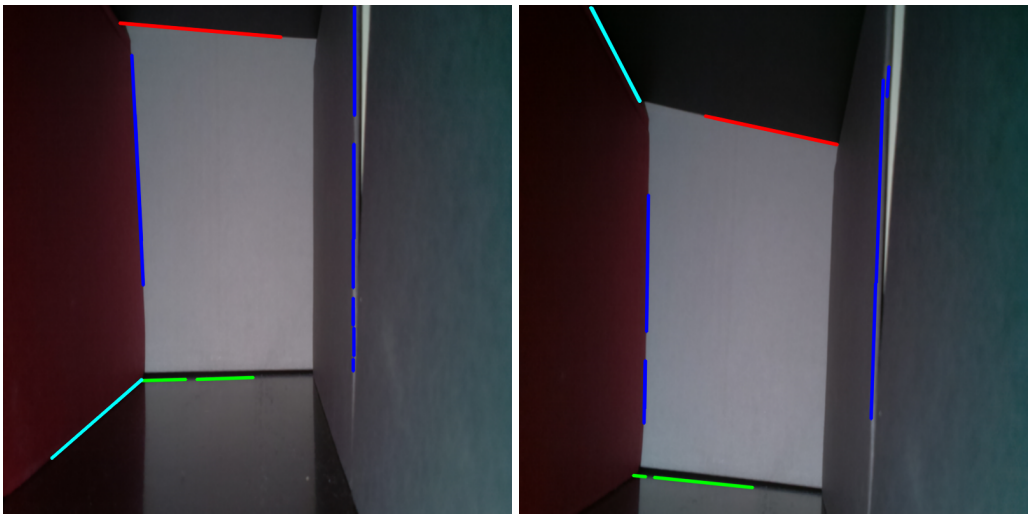


Figura 8.68 – Error por diagonales insuficientes

En la Figura 8.68, se obtienen dos casos consecutivos en los que la ausencia de más de una diagonal detectada en la imagen impide delimitar las superficies.

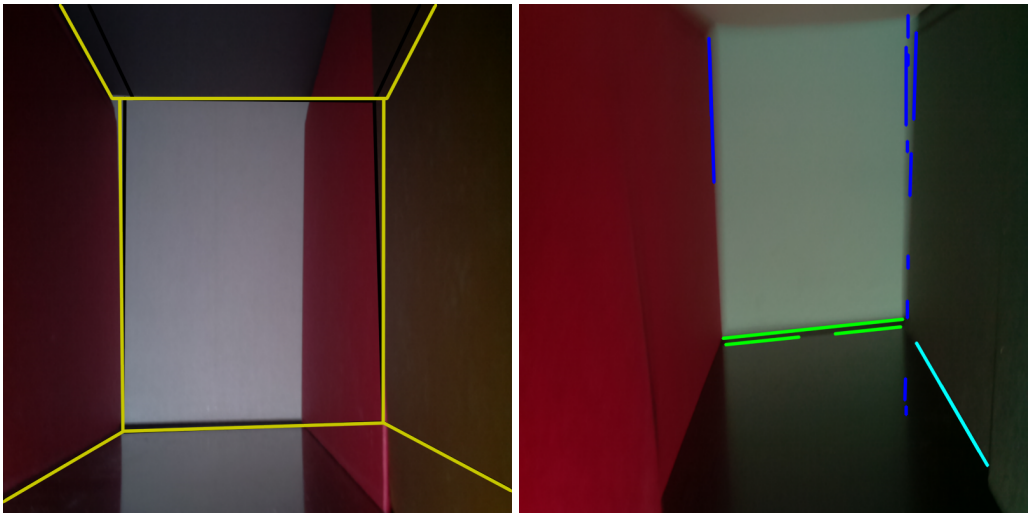


Figura 8.69 – Error del programa y error por desenfoque

En la Figura 8.69, se dispone de los mismos colores en pared y techo que en el caso anterior pero, al ubicar la cámara en una posición válida y aumentar la iluminación, solo el límite derecho estará mal ubicado. Al haber una línea vertical que cambie el color de forma brusca, este será un error del programa diseñado y no de las condiciones de la cámara.

En la segunda imagen de esa misma Figura, se busca provocar un fallo propio de la práctica real. Tras cambiar el color del techo y colocar uno similar al de la pared, se obtiene la imagen mientras se desplaza ligeramente la cámara en el momento de la toma de la fotografía, pudiendo ver así como afecta la borrosidad al programa. El resultado es que el límite de color similar(techo-pared frontal) no es detectado debido al desenfoque de la cámara. Las diagonales serán también insuficientes para delimitar el cuarto.

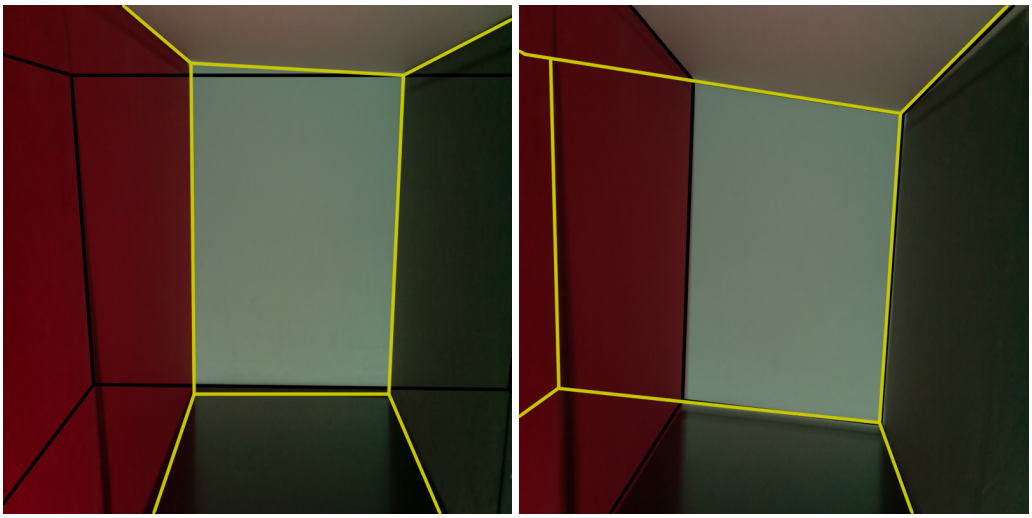


Figura 8.70 – Secuencia del programa de realidad aumentada 1

Las Figuras 8.70 y 8.71 han sido obtenidas con una misma ejecución del programa. En este caso, la cámara está ubicada correctamente y las condiciones de iluminación son favorables. De las cuatro imágenes, en la primera y en la cuarta serán los límites obtenidos a color los que funcionan perfecto, mientras que en la segunda será el límite obtenido en escala de grises. Se demuestra así que, dependiendo del tratamiento inicial de la imagen se obtienen resultados distintos y que, ambos métodos, son necesarios para obtener un mayor número de aciertos.

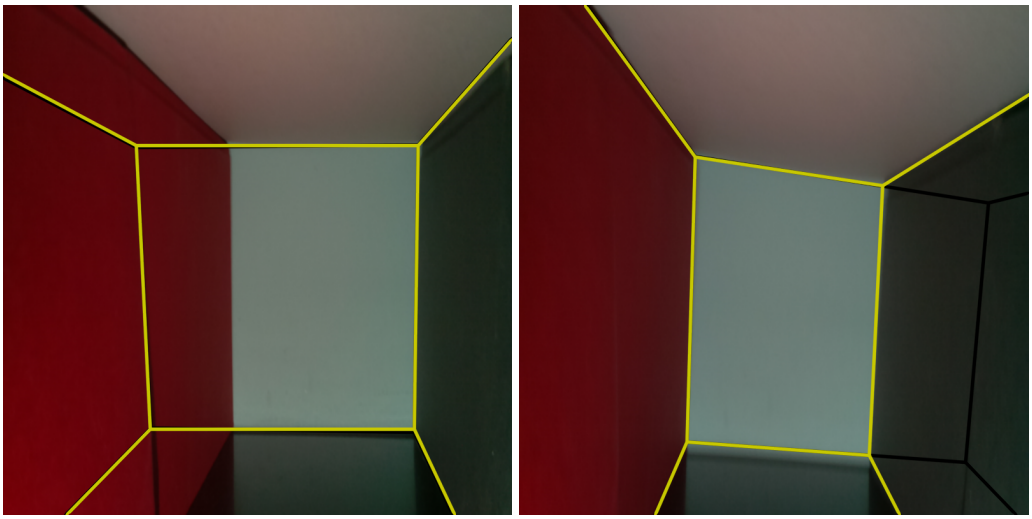


Figura 8.71 – Secuencia del programa de realidad aumentada 2

Tras probar el sistema de delimitación, se considera que los resultados obtenidos permiten llevar a la práctica la variación del programa con cambio de textura. A continuación se muestran tres secuencias, de dos imágenes cada una, con los resultados obtenidos:

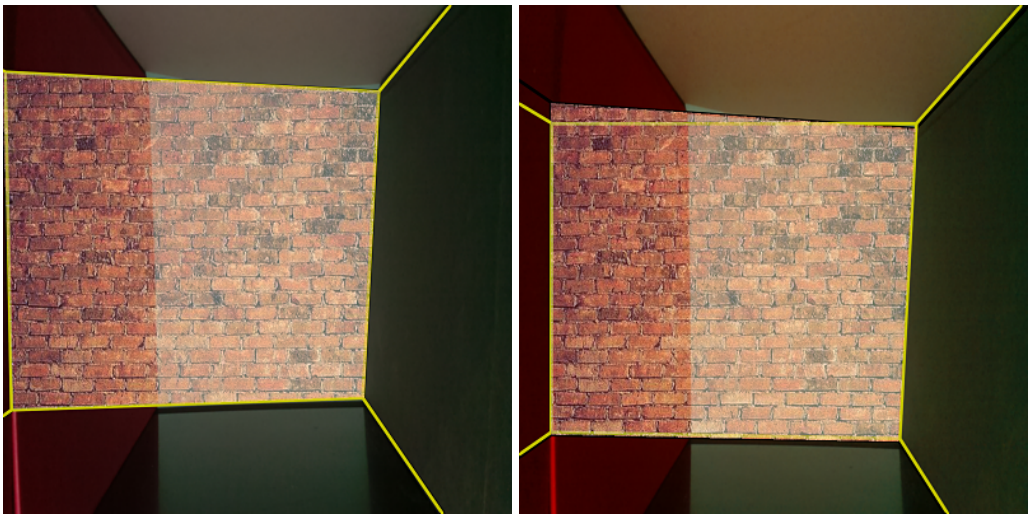


Figura 8.72 – Secuencia 1 del programa de realidad aumentada de cambio de pared

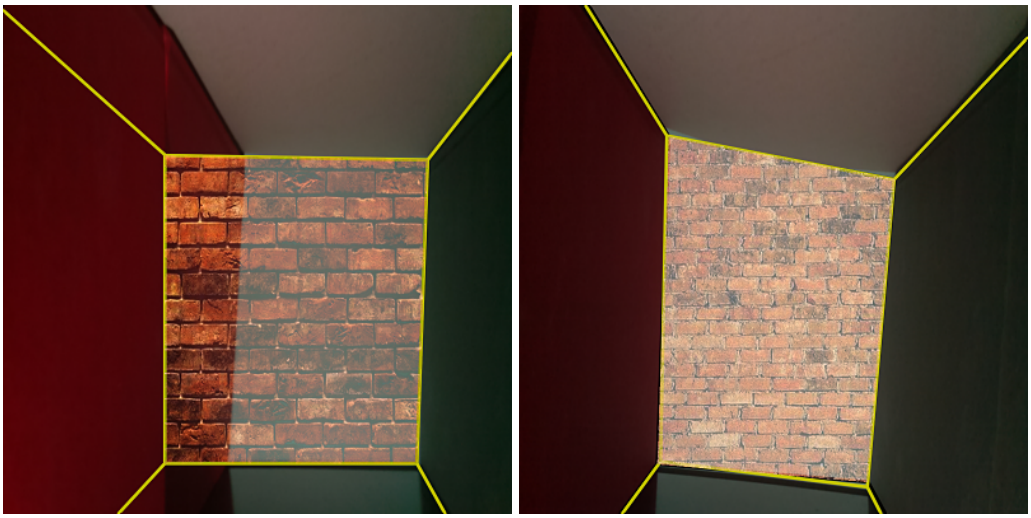


Figura 8.73 – Secuencia 2 del programa de realidad aumentada de cambio de pared

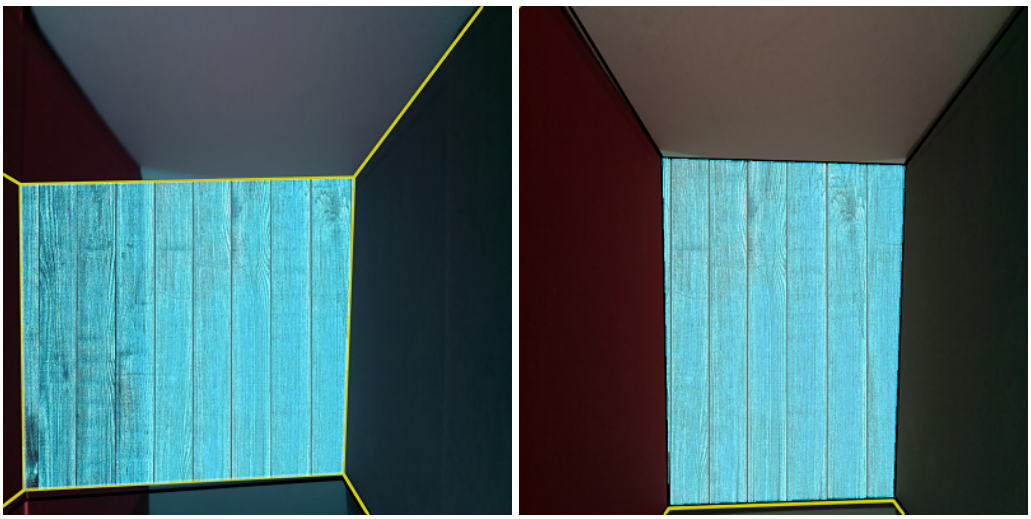


Figura 8.74 – Secuencia 3 del programa de realidad aumentada de cambio de pared

El programa se verá muy ralentizado al añadir el cambio de textura, a pesar de ello, logra detectar las superficies, o al menos detectar tres de los cuatro límites, la mitad de las veces que es ejecutado de forma práctica.

9 ORDEN DE PRIORIDAD ENTRE LOS DOCUMENTOS

1. Presupuesto
2. Memoria
3. Anexos

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

ANEXOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

Índice del documento ANEXOS

10 DOCUMENTACIÓN DE PARTIDA

10.1. Propuesta inicial de asignación del TFG

Escuela Universitaria Politécnica Ferrol
UNIVERSIDADE DA CORUÑA

Intranet eup

ESCUELA UNIVERSITARIA POLITÉCNICA

ASIGNACIÓN DE TRABAJO FIN DE GRADO

En virtud de la solicitud efectuada por:
En virtud da solicitude efectuada por:

APELLIDOS, NOMBRE: *Palacios Fragoso,Manuel*
APELIDOS E NOME:

DNI: [redacted] **Fecha de Solicitud:** Feb2019
DNI: *Fecha de Solicitude:*

Alumno de esta escuela en la titulación de Grado en Ingeniería en Electrónica Industrial y Automática, se le comunica que la Comisión de Proyectos ha decidido asignarle el siguiente Trabajo Fin de Grado:

O alumno de esta escola na titulación de Grado en Enxeñería en Electrónica Industrial e Automática, comunícaselle que a Comisión de Proxectos ha decidido asignarlle o seguinte Traballo Fin de Grado:

Título T.F.G:Implementación de un sistema de reconocimiento y delimitación de superficies.

Número TFG: *770G01A175*

TUTOR:(Titor) *Jove Pérez,Esteban*

COTUTOR/CODIRECTOR: *Héctor Quintián Pardo*

La descripción y objetivos del Trabajo son los que figuran en el reverso de este documento:

A descrición e obxectivos do proxecto son os que figuran no reverso deste documento.

Ferrol a Viernes, 1 de Marzo del 2019

Retirei o meu Traballo Fin de Grado o dia _____ de _____ do ano _____

Fdo: Palacios Fragoso,Manuel

DESCRIPCIÓN Y OBJETIVO:OBJETO:
En este Trabajo Final de Grado se realizará el estudio y la implementación de un sistema de reconocimiento de superficies mediante el uso de una cámara digital y librerías de tratamiento de imágenes. Inicialmente, se hará un estudio del tipo de hardware y software apropiado para aplicar las librerías de tratamiento de imágenes para el reconocimiento de superficies. Posteriormente, tratará de implementarse el reconocimiento y delimitación de superficies en un entorno cerrado. Se estudiará la posibilidad de utilizar el reconocimiento de superficies tipo pared, suelo y techo para aplicar técnicas de realidad aumentada.

ALCANCE:
Análisis del hardware y software adecuado para el uso de reconocimiento y delimitación de superficies.

Aplicación de librerías de tratamiento de imágenes para el entorno seleccionado sobre diversas superficies tipo en un entorno cerrado.

Estudio de la posibilidad de emplear el sistema de reconocimiento y delimitación de superficies en aplicaciones de realidad aumentada.

11 CÓDIGO DEL PROGRAMA FINAL

```
import cv2
from scipy import stats
import numpy as np
import time
import math
import sys
import math
import picamera

q='si'
destruir=0
color=0
while q=='si':
    with picamera.PiCamera() as picam:
        picam.capture('/home/pi/Desktop/imagenestfg/foto.jpg')
        picam.close()

    imgoriginal = cv2.imread('/home/pi/Desktop/imagenestfg/foto.jpg')
    t_img=len(imgoriginal)
    imgoriginal=cv2.resize(imgoriginal,(800,800))

    #Se filtra el ruido de la imagen
    #por el tamaño de la imagen se distingue la procedencia de la foto
    if t_img>900 and t_img<=1500:
        img=cv2.medianBlur(imgoriginal,5)

        img= cv2.GaussianBlur(img, (3,3), 0)
    else:
        img= cv2.GaussianBlur(imgoriginal, (3,3), 0)

    #se obtiene la imagen en escala de grises
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #edges1 se obtiene los bordes en escala de grises y edges2 a color
    if t_img<=900:
        edges1 = cv2.Canny(gray, 10,20, apertureSize = 3)
        edges2 = cv2.Canny(img, 10,30, apertureSize = 3)

    if t_img<=1500 and t_img>900:
        edges1 = cv2.Canny(gray, 10,40, apertureSize = 3)
        edges2 = cv2.Canny(img, 20,40, apertureSize = 3)

    if t_img>1500:
        edges1 = cv2.Canny(gray, 40,80, apertureSize = 3)
```

```
45          edges2 = cv2.Canny(img, 40,70, apertureSize = 3)

47      #algunas se detectan mejor con valores asignados exclusivamente para ellas
        #room3,4,5,7
49      #if t_img<=1500 and t_img>900:
        #    edges1 = cv2.Canny(gray, 10,40, apertureSize = 3)
51      #    edges2 = cv2.Canny(img, 20,40, apertureSize = 3)
        #room8:
53      #if t_img<=1500 and t_img>900:
        #     edges1 = cv2.Canny(gray, 20,50, apertureSize = 3)
55      #    edges2 = cv2.Canny(img, 10,50, apertureSize = 3)

57      #sedetectan líneas en ambas imagenes con HoughlinesP
        lines1 = cv2.HoughLinesP(edges1, 1, np.pi/180, 100, minLineLength=10, maxLineGap
        =100)
59      lines2 = cv2.HoughLinesP(edges2, 1, np.pi/180, 100, minLineLength=10, maxLineGap
        =100)

61      #creo matriz de unos para rellenar con datos
        ncolum1=len(lines1) #numero de filas
63      ncolum2=len(lines2)
        #matrices grises
65      ones1=np.ones((ncolum1,4))
        ones1=np.array(ones1)
67      ones2=np.ones((ncolum1,4))
        ones2=np.array(ones2)
69      ones3=np.ones((ncolum1,4))
        ones3=np.array(ones3)
71      ones4=np.ones((ncolum1,4))
        ones4=np.array(ones4)
73      ones5=np.ones((ncolum1,4))
        ones5=np.array(ones5)
75      ones6=np.ones((ncolum1,4))
        ones6=np.array(ones6)
77      ones7=np.ones((ncolum1,4))
        ones7=np.array(ones7)
79      ones8=np.ones((ncolum1,4))
        ones8=np.array(ones8)
81
        #matrices con color
83      ones9=np.ones((ncolum2,4))
        ones9=np.array(ones9)
85      ones10=np.ones((ncolum2,4))
        ones10=np.array(ones10)
87      ones11=np.ones((ncolum2,4))
        ones11=np.array(ones11)
89      ones12=np.ones((ncolum2,4))
        ones12=np.array(ones12)
91      ones13=np.ones((ncolum2,4))
        ones13=np.array(ones13)
```

```
    ones14=np.ones((ncolum2,4))
    ones14=np.array(ones14)
    ones15=np.ones((ncolum2,4))
    ones15=np.array(ones15)
    ones16=np.ones((ncolum2,4))
    ones16=np.array(ones16)

    #FUNCION CREAR LÍNEAS----------

    def trazalineas(matriz,limiteyinf,limiteysup,limitexinf,limitexsup,pendientemin1
    ,pendientemax1,pendientemin2,pendientemax2,orientacion,color):
        nfila=0
        filaexceso=0
        #segun si se trabaja a color o en grises, se selecciona una matriz de lineas
        if color==0:
            lines=lines1
            ncolum1=len(lines1)
        elif color==1:
            lines=lines2
            ncolum1=len(lines2)
        #arrays a uno para pendiente y proyeccion
        mpre=np.ones((ncolum1,1))
        ypre=np.ones((ncolum1,1))

        #ESTE BLOQUE CALCULA TODAS LAS PENDIENTES DE LAS LINEAS HALLADAS

        for line in lines:
            #segun si la linea es vertical u horizontal, se alternan los paraámetros
            #para evitar divisiones por cero
            if orientacion==0:
                x1, y1, x2, y2 = line[0]
            elif orientacion==1:
                y1 ,x1, y2, x2 = line[0]
            #se delimita la zona de busqueda de la línea
            if y1>=limiteyinf and y1<=limiteysup and x1>=limitexinf and x1<=
    limitexsup:
                if x1!=x2:
                    #calculo la pendiente
                    m=(y2-y1)/(x2-x1)

                    #redondeo pendiente
                    m=round(m/0.005)*0.005

                    if (m<pendientemax1 and m>pendientemin1) or (m>pendientemin2 and
     m<pendientemax2):
                        #se añade la pendiente al vector si está dentro de los pará
    metros establecidos
                        mpre[nfila]=m

                        nfila=nfila+1
```

```
                    else:
                        filaexceso=filaexceso+1
                else:
                    filaexceso=filaexceso+1
            else:
                    filaexceso=filaexceso+1

        #quitar datos sobrantes del vector de pendientes
        borrardesdefila=ncolum1-filaexceso
        mpre=mpre[0:borrardesdefila][:]

        #se realiza la moda del vector de pendientes
        mpre,count= stats.mode(mpre)

        #ESTE BLOQUE SELECCIONA LAS LINEAS CON LA PENDIENTE MÁS REPETIDA
        nfila=0
        filaexceso=0

        for line in lines:
            if orientacion==0:
                x1, y1, x2, y2 = line[0]
            elif orientacion==1:
                y1 ,x1, y2, x2 = line[0]

            if y1>=limiteyinf and y1<=limiteysup and x1>=limitexinf and x1<=
    limitexsup:
                if x1!=x2:
                    m=(y2-y1)/(x2-x1)
                    m=round(m/0.005)*0.005

                    if m==mpre:
                        x1=round(x1)
                        y1=round(y1)
                        x2=round(x2)
                        y2=round(y2)

                        #se guardan las lineas con la pendiente más
                        if orientacion==0:
                            matriz[nfila]=x1,y1,x2,y2
                        elif orientacion==1:
                            matriz[nfila]=y1,x1,y2,x2
                        nfila=nfila+1
                    else:
                        filaexceso=filaexceso+1
                else:
                    filaexceso=filaexceso+1
            else:
                filaexceso=filaexceso+1

        #borrar filas sobrantes
```

```
        borrardesdefila=ncolum1-filaexceso
189     matriz=matriz[0:borrardesdefila][:]

191     #se prepara el vector de proyecciones en función de la cantidad de líneas
    que quedan
        ypre=np.ones((len(matriz),1))
193
        #ESTE BLOQUE CALCULA TODAS LAS PROYECCIONES DE LAS LINEAS RESTANTES
195     fila=0
        nfila=0
197     filaexceso=0
        for fila in range(0,len(matriz)):
199         if orientacion==0:
                x1, y1, x2, y2 = matriz[fila]
201         elif orientacion==1:
                y1 ,x1, y2, x2 = matriz[fila]
203         #quitamos las lineas verticales por problemas de calculo. Primero nos
    centramos en lineas de suelo y techo
            if y1>=limiteyinf and y1<=limiteysup and x1>=limitexinf and x1<=
    limitexsup:
205             if x1!=x2:
                    #se calcula ahora la proyección de cada línea
207                 m=(y2-y1)/(x2-x1)
                    ptoinicio=+y1-m*x1
209                 #redondeo pendiente y proyeccion
                    m=round(m/0.005)*0.005
211                 ptoinicio=round(ptoinicio/5)*5
                    #coloco en columna
213                 ypre[nfila]=ptoinicio
                    nfila=nfila+1
215             else:
                    filaexceso=filaexceso+1
217         else:
                filaexceso=filaexceso+1
219
        #quitar datos sobrantes
221     borrardesdefila=ncolum1-filaexceso
        mpre=mpre[0:borrardesdefila][:]
223     ypre=ypre[0:borrardesdefila][:]
        ypre,count= stats.mode(ypre)
225     #descomentar para ver la proyección y la pendiente usada
        #print(mpre)
227     #print(ypre)

229     #ESTE BLOQUE SELECCIONA LAS LÍNEAS CON LA PROYECCIÓN MÁS REPETIDA
        fila=0
231     nfila=0
        filaexceso=0
233
        for fila in range(0,len(matriz)):
```

```
            if orientacion==0:
                x1, y1, x2, y2 = matriz[fila]
            elif orientacion==1:
                y1 ,x1, y2, x2 = matriz[fila]

            #quitamos las lineas verticales por problemas de calculo. Primero nos centramos en lineas de suelo y techo
            if x1!=x2:

                m=(y2-y1)/(x2-x1)
                m=round(m/0.001)*0.001

                ptoinicio=y1-m*x1
                ptoinicio=round(ptoinicio/5)*5

                if ptoinicio==ypre:
                    x1=round(x1)
                    y1=round(y1)
                    x2=round(x2)
                    y2=round(y2)
                    if orientacion==0:
                        matriz[nfila]=x1,y1,x2,y2
                    elif orientacion==1:
                        matriz[nfila]=y1,x1,y2,x2

                    nfila=nfila+1
                else:
                    filaexceso=filaexceso+1
            else:
                filaexceso=filaexceso+1
        #borrar filas sobrantes
        borrardesdefila=len(matriz)-filaexceso
        matriz=matriz[0:borrardesdefila][:]
        return matriz

#A CONTINUACIÓN SE MUESTRA LA FUNCION CONFIGURADA PARA LA BUSQUEDA DE CADA UNA DE LAS LÍNEAS
    #EN BLANCO Y NEGRO----------------------------------

    #-SUELO-PARED
    ones1=trazalineas(matriz=ones1,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=0,color=0)
    #TECHO-PARED
    ones2=trazalineas(matriz=ones2,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=0,color=0)
    #PARED PARED IZQUIERDA
    ones3=trazalineas(matriz=ones3,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
```

```
    pendientemax2=0.2,orientacion=1,color=0)
    #PARED PARED DERECHA
279 ones4=trazalineas(matriz=ones4,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=1,color=0)
    #DIAGONAL SUPERIOR IZQUIERDA
281 ones5=trazalineas(matriz=ones5,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=300,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=1,color=0)
    #DIAGONAL INFERIOR DERECHA
283 ones6=trazalineas(matriz=ones6,limiteyinf=500,limiteysup=800,limitexinf=500,
    limitexsup=800,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=0)
    #DIAGONAL INFERIOR IZQUIERDA
285 ones7=trazalineas(matriz=ones7,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=300,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=0)
    #DIAGONAL SUPERIOR DERECHA
287 ones8=trazalineas(matriz=ones8,limiteyinf=0,limiteysup=300,limitexinf=500,
    limitexsup=800,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=0)

289 #A COLOR----------------------------------

291 #SUELO-PARED
    ones9=trazalineas(matriz=ones9,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=0,color=1)
293 #TECHO-PARED
    ones10=trazalineas(matriz=ones10,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=0,color=1)
295 #PARED PARED IZQUIERDA
    ones11=trazalineas(matriz=ones11,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=1,color=1)
297 #PARED PARED DERECHA
    ones12=trazalineas(matriz=ones12,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=800,pendientemin1=-0.2,pendientemax1=0.2,pendientemin2=-0.2,
    pendientemax2=0.2,orientacion=1,color=1)
299 #DIAGONAL SUPERIOR IZQUIERDA
    ones13=trazalineas(matriz=ones13,limiteyinf=0,limiteysup=300,limitexinf=0,
    limitexsup=300,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=1,color=1)
301 #DIAGONAL INFERIOR DERECHA
    ones14=trazalineas(matriz=ones14,limiteyinf=500,limiteysup=800,limitexinf=500,
    limitexsup=800,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=1)
303 #DIAGONAL INFERIOR IZQUIERDA
```

```
    ones15=trazalineas(matriz=ones15,limiteyinf=500,limiteysup=800,limitexinf=0,
    limitexsup=300,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=1)
    #DIAGONAL SUPERIOR DERECHA
    ones16=trazalineas(matriz=ones16,limiteyinf=0,limiteysup=300,limitexinf=500,
    limitexsup=800,pendientemin1=0.5,pendientemax1=2,pendientemin2=-2,pendientemax2
    =-0.5,orientacion=0,color=1)

    #LOCALIZACION PUNTO FUGA-----------------------------
    #print(type(ones1))
    #print(ones5[0])
    nfila=0
    i=0
    #se crea una matriz lo bastante grande para almacenar las líneas de punto de
    fuga
    fuga_todos=np.ones((256,4))
    filaexceso=0
    comparaciondiagonal=1,1,1,1

    #para el punto de fuga habrá que emplear dos diagonales
    for i in range(0,7):
        diagonal1=1, 1, 1, 1
        diagonal1=np.array(diagonal1)
        #se busca una diagonal que haya sido detectada
        if i==0 and len(ones5)!=0:
            diagonal1=ones5[0][:]
        elif i==1 and len(ones6)!=0:
            diagonal1=ones6[0][:]
        elif i==2 and len(ones7)!=0:
            diagonal1=ones7[0][:]
        elif i==3 and len(ones8)!=0:
            diagonal1=ones8[0][:]
        elif i==4 and len(ones13)!=0:
            diagonal1=ones13[0][:]
        elif i==5 and len(ones14)!=0:
            diagonal1=ones14[0][:]
        elif i==6 and len(ones15)!=0:
            diagonal1=ones15[0][:]
        elif i==7 and len(ones16)!=0:
            diagonal1=ones16[0][:]

        if len(diagonal1)!=0:
            #una vez se ha encontrado una diagonal detectada se obtienen su
    pendiente y su proyección
            x1,y1,x2,y2,=diagonal1
            #print(x1)
            x11=int(x1)
            y11=int(y1)
            x12=int(x2)
            y12=int(y2)
```

```
            if x11!=x12 and y11!=y12:
349             m1=(y12-y11)/(x12-x11)
                ptoinicio1=+y11-m1*x11
351             j=0
                #se busca ahora la segunda diagonal. Esta no debe estar ni en el
    mismo cuadrante ni en el opuesto.
353             for j in range(0,7):
                    diagonal2=1, 1, 1, 1
355                 diagonal2=np.array(diagonal2)

357                 if j==0 and i!=4 and i!=0 and len(ones5)!=0:
                        diagonal2=ones5[0][:]
359                 elif j==1 and i!=5 and i!=1 and len(ones6)!=0:
                        diagonal2=ones6[0][:]
361                 elif j==2 and i!=6 and i!=2  and len(ones7)!=0:
                        diagonal2=ones7[0][:]
363                 elif j==3 and i!=7 and i!=3  and len(ones8)!=0:
                        diagonal2=ones8[0][:]
365                 elif j==4 and i!=0 and i!=4  and len(ones13)!=0:
                        diagonal2=ones13[0][:]
367                 elif j==5 and i!=1 and i!=5  and len(ones14)!=0:
                        diagonal2=ones14[0][:]
369                 elif j==6 and i!=2 and i!=6  and len(ones15)!=0:
                        diagonal2=ones15[0][:]
371                 elif j==7 and i!=3 and i!=7  and len(ones16)!=0:
                        diagonal2=ones16[0][:]
373
                    if len(diagonal2)!=0:
375                     #se comprueba que no es una diagonal falsa solo con unos
                        if not (diagonal2 == comparaciondiagonal).all():
377                         x1=diagonal2[0]
                            y1=diagonal2[1]
379                         x2=diagonal2[2]
                            y2=diagonal2[3]
381                         x1=int(x1)
                            y1=int(y1)
383                         x2=int(x2)
                            y2=int(y2)
385
                            if x1!=x2 and y1!=y2:
387                             m2=(y2-y1)/(x2-x1)
                                ptoinicio2=+y1-m2*x1
389
                                #se calcula el cruce entre lineas
391                             if m1!=m2:
                                    xfuga=(ptoinicio2-ptoinicio1)/(m1-m2)
393                                 yfuga=m2*xfuga+ptoinicio2
                                    #se comprueba que esté dentro de la imagen
395                                 if xfuga>0 and xfuga<800 and yfuga>0 and yfuga
    <800:
```

```
                                    #se redondea la ubicación del punto de fuga
397                                 xfuga=round(xfuga/5)*5
                                    yfuga=round(yfuga/5)*5
399                                 #se guardan los puntos de fuga y la
    combinación de diagonales
                                    fuga_todos[nfila]=xfuga, yfuga,i,j
401                                 nfila=nfila+1

403     if len(fuga_todos)!=0:
            fuga_todos=fuga_todos[0:nfila][:]
405         #transpuesta para buscar moda de la coordenada x del punto de fuga
            fuga_todos=np.transpose(fuga_todos)
407         xfugarep,count= stats.mode(fuga_todos[0])
            fuga_todos=np.transpose(fuga_todos)
409     #print(xfugarep)

411     if len(fuga_todos)!=0:
            i=0
413         nfila=0
            for i in range(0,len(fuga_todos)):
415             x,y,id_diag1,id_diag2=fuga_todos[i][:]
                if x==xfugarep:
417
                    fuga_todos[nfila]=x,y,id_diag1,id_diag2
419                 nfila=nfila+1
            #de los puntos de fuga con x más repetido, se busca los que más repiten
421         fuga_todos=fuga_todos[0:nfila][:]
            nfila=0
423         fuga_todos=np.transpose(fuga_todos)
            yfugarep,count= stats.mode(fuga_todos[1][:])
425         fuga_todos=np.transpose(fuga_todos)

427     if len(fuga_todos)!=0:
            i=0
429         nfila=0
            for i in range(0,len(fuga_todos)):
431             x,y,id_diag1,id_diag2=fuga_todos[i][:]
                if y==yfugarep:
433                 fuga_todos[nfila]=x,y,id_diag1,id_diag2
                    nfila=nfila+1
435         #punto de fuga final y diagonales que lo cumplen
            fuga_todos=fuga_todos[0][:]
437
        #REDIMENSIONADO DE LÍNEAS ————————————————————————
439     #segun las diagonaales usadas para hallar el pto fuga se delimita primero una
        #linea u otra. Se prescindirá del resto de diagonales halladas.
441
        #id 0 y 4 diag sup izq
443     #id 1 y 5 diag inf dcha
        #id 2 y 6 diag inf izq
```

```
445     #id 3 y 7 diag sup dcha

447     #en función del identificador de las diagonales usadas, son selecionadas

449     diagonal1=1,1,1,1
        diagonal2=1,1,1,1
451     if len(fuga_todos)!=0:
            if id_diag1==0:
453             diagonal1=ones5[0][:]
            elif id_diag1==1:
455             diagonal1=ones6[0][:]
            elif id_diag1==2:
457             diagonal1=ones7[0][:]
            elif id_diag1==3:
459             diagonal1=ones8[0][:]
            elif id_diag1==4:
461             diagonal1=ones13[0][:]
            elif id_diag1==5:
463             diagonal1=ones14[0][:]
            elif id_diag1==6:
465             diagonal1=ones15[0][:]
            elif id_diag1==7:
467             diagonal1=ones16[0][:]

469         if id_diag2==0:
                diagonal2=ones5[0][:]
471         elif id_diag2==1:
                diagonal2=ones6[0][:]
473         elif id_diag2==2:
                diagonal2=ones7[0][:]
475         elif id_diag2==3:
                diagonal2=ones8[0][:]
477         elif id_diag2==4:
                diagonal2=ones13[0][:]
479         elif id_diag2==5:
                diagonal2=ones14[0][:]
481         elif id_diag2==6:
                diagonal2=ones15[0][:]
483         elif id_diag2==7:
                diagonal2=ones16[0][:]
485
#FUNCIÓNES DE REDIMENSIONADO----------------------------
487     def redimensionhorizontal(lineadcha,lineaizq,lineacortada):
            if len(lineadcha)!=0 and len(lineaizq)!=0 and len(lineacortada)!=0:
489             #se guardan los valores de la línea que corta por la izquierda y la que
    corta por la derecha
                xc11,yc11,xc12,yc12=lineadcha
491             xc21,yc21,xc22,yc22=lineaizq
                #se guardan los valores de la línea que será redimensionada
493             x1,y1,x2,y2=lineacortada #linea de suelo o de techo
```

```
            #Al haber una linea en cada orientacion, no sirve cambiar el orden
495         #de los parametros para evitar el problema de 2 ptos en una x.
            #Como normalmente redondeo, no supone un cambio significativo sumar
    incluso 1 pixel cuando son iguales.
497
            if x2==x1:
499             x2=x2+1
            if xc11==xc12:
501             xc12=xc12+1
            if xc21==xc22:
503             xc22=xc22+1

505         mcortada=(y2-y1)/(x2-x1)
            m1=(yc12-yc11)/(xc12-xc11)
507         m2=(yc22-yc21)/(xc22-xc21)

509         ptoiniciocortada=+y1-mcortada*x1
            ptoinicio1=+yc11-m1*xc11
511         ptoinicio2=+yc21-m2*xc21

513         if xc21<300 or xc22<300:
                if mcortada==m2:
515                 m2=m2+1
                xcorte=(ptoiniciocortada-ptoinicio2)/(m2-mcortada)
517             ycorte=mcortada*xcorte+ptoiniciocortada
                #se sustituye el punto más a la izquierda de la línea por el punto
    de corte
519             x1=xcorte
                y1=ycorte
521
            if xc11>500 or xc12>500:
523             if mcortada==m1:
                    m1=m1+1
525             xcorte=(ptoiniciocortada-ptoinicio1)/(m1-mcortada)
                ycorte=mcortada*xcorte+ptoiniciocortada
527             #se sustituye el punto más a la derecha de la línea por el punto de
    corte
                x2=xcorte
529             y2=ycorte

531         x1=round(x1)
            y1=round(y1)
533         x2=round(x2)
            y2=round(y2)
535
            lineah=x1,y1,x2,y2
537         #print(lineah)
        return lineah
539
    def redimensionvertical(lineainf,lineasup,lineacortada):
```

```
541
        if len(lineainf)!=0 and len(lineasup)!=0 and len(lineacortada)!=0:
543             xc11,yc11,xc12,yc12=lineainf
                xc21,yc21,xc22,yc22=lineasup
545             x1,y1,x2,y2=lineacortada #linea de suelo o de techo

547             if x2==x1:
                    x2=x2+0.25
549             if xc11==xc12:
                    x12=x12+0.25
551             if xc21==xc22:
                    xc22=xc22+0.25
553
                mcortada=(y2-y1)/(x2-x1)
555             m1=(yc12-yc11)/(xc12-xc11)
                m2=(yc22-yc21)/(xc22-xc21)
557
                ptoiniciocortada=+y1-mcortada*x1
559             ptoinicio1=+yc11-m1*xc11
                ptoinicio2=+yc21-m2*xc21
561
                if yc21<300 or yc22<300:
563                 xcorte=(ptoiniciocortada-ptoinicio2)/(m2-mcortada)
                    ycorte=mcortada*xcorte+ptoiniciocortada
565                 #se sustituye el punto superior de la línea por el punto de corte
                    x1=xcorte
567                 y1=ycorte

569             if yc11>500 or yc12>500:
                    xcorte=(ptoiniciocortada-ptoinicio1)/(m1-mcortada)
571                 ycorte=mcortada*xcorte+ptoiniciocortada
                    #se sustituye el punto inferior de la línea por el punto de corte
573                 x2=xcorte
                    y2=ycorte
575
                x1=round(x1)
577             y1=round(y1)
                x2=round(x2)
579             y2=round(y2)

581             lineav=x1,y1,x2,y2
                #print(lineav)
583         return lineav

585     sueloprueba=np.ones((1,4))
        techoprueba=np.ones((1,4))
587     paredizquierda=np.ones((1,4))
        paredderecha=np.ones((1,4))
589     sueloprueba2=np.ones((1,4))
        techoprueba2=np.ones((1,4))
```

```
    paredizquierda2=np.ones((1,4))
    paredderecha2=np.ones((1,4))

    #redimensionado de cada línea
    if len(ones4)!=0 and len(ones3)!=0 and len(ones1)!=0:
        sueloprueba[0]=redimensionhorizontal(ones4[0][:],ones3[0][:],ones1[0][:])
    if len(ones4)!=0 and len(ones3)!=0 and len(ones2)!=0:
        techoprueba[0]=redimensionhorizontal(ones4[0][:],ones3[0][:],ones2[0][:])
    if len(ones12)!=0 and len(ones11)!=0 and len(ones9)!=0:
     sueloprueba2[0]=redimensionhorizontal(ones12[0][:],ones11[0][:],ones9[0])
    if len(ones4)!=0 and len(ones11)!=0 and len(ones10)!=0:
      techoprueba2[0]=redimensionhorizontal(ones12[0][:],ones11[0][:],ones10[0][:])

    if len(ones1)!=0 and len(ones2)!=0 and len(ones3)!=0:
        paredizquierda[0]=redimensionvertical(ones1[0][:],ones2[0][:],ones3[0][:])
    if len(ones1)!=0 and len(ones2)!=0 and len(ones4)!=0:
        paredderecha[0]=redimensionvertical(ones1[0][:],ones2[0][:],ones4[0][:])
    if len(ones9)!=0 and len(ones10)!=0 and len(ones11)!=0:
        paredizquierda2[0]=redimensionvertical(ones9[0][:],ones10[0][:],ones11
    [0][:])
    if len(ones9)!=0 and len(ones10)!=0 and len(ones12)!=0:
        paredderecha2[0]=redimensionvertical(ones9[0][:],ones10[0][:],ones12[0][:])

    #NUEVO TRAZADO DE DIAGONALES
    ____________________________________________________________

    #usando el punto de fuga se pueden trazar nuevas diagonales acorde al recuadro
    def trazadiagonales(ptofuga,ptocorte,tipo):
        ndiag=np.ones((1,4))
        ndiag0=np.ones((1,4))

        xcorte1=1
        xcorte2=1
        ycorte1=1
        ycorte2=1

        xc1,yc1,xc2,yc2=ptocorte[0]
        xf,yf,n1,n2=ptofuga
        if tipo==1:
            if xf!=xc1:
                m=(yf-yc1)/(xf-xc1)
                ptoinicio=+yf-m*xf
                ycorte1=800
                xcorte1=(ycorte1-ptoinicio)/m

            if xf!=xc2:
                m=(yf-yc2)/(xf-xc2)
                ptoinicio=+yf-m*xf
                ycorte2=800
                xcorte2=(ycorte2-ptoinicio)/m
```

```
639         if tipo==2:
                if xf!=xc1:
641                 m=(yf-yc1)/(xf-xc1)
                    ptoinicio=+yf-m*xf
643                 ycorte1=0
                    xcorte1=(ycorte1-ptoinicio)/m
645
                if xf!=xc2:
647                 m=(yf-yc2)/(xf-xc2)
                    ptoinicio=+yf-m*xf
649                 ycorte2=0
                    xcorte2=(ycorte2-ptoinicio)/m
651         xcorte1=round(xcorte1)
            ycorte1=round(ycorte1)
653         xcorte2=round(xcorte2)
            ycorte2=round(ycorte2)
655
            ndiag=xc1,yc1,xcorte1,ycorte1
657         ndiag0=xc2,yc2,xcorte2,ycorte2
            return ndiag,ndiag0
659
    ndiag1=1,1,1,1
661 ndiag2=1,1,1,1
    ndiag3=1,1,1,1
663 ndiag4=1,1,1,1
    ndiag8=1,1,1,1
665 ndiag9=1,1,1,1
    ndiag10=1,1,1,1
667 ndiag11=1,1,1,1

669 #trazado de diagonales
    if len(sueloprueba)!=0 and not (sueloprueba == ndiag9).all() and len(fuga_todos)
    !=0:
671     ndiag1,ndiag2=trazadiagonales(ptofuga=fuga_todos,ptocorte=sueloprueba,tipo
    =1)
    if len(techoprueba)!=0 and not (techoprueba == ndiag9).all() and len(fuga_todos)
    !=0:
673     ndiag3,ndiag4=trazadiagonales(ptofuga=fuga_todos,ptocorte=techoprueba,tipo
    =2)
    if len(sueloprueba2)!=0 and not (sueloprueba2 == ndiag9).all() and len(
    fuga_todos)!=0:
675     ndiag8,ndiag9=trazadiagonales(ptofuga=fuga_todos,ptocorte=sueloprueba2,tipo
    =1)
    if len(techoprueba2)!=0 and not (techoprueba2 == ndiag9).all() and len(
    fuga_todos)!=0:
677     ndiag10,ndiag11=trazadiagonales(ptofuga=fuga_todos,ptocorte=techoprueba2,
    tipo=2)

679 #se mete en corchetes para que estén en el mismo formato que las otras líneas al
     dibujarlas
```

```
    ndiag1=[ndiag1]
    ndiag2=[ndiag2]
    ndiag3=[ndiag3]
    ndiag4=[ndiag4]
    ndiag8=[ndiag8]
    ndiag9=[ndiag9]
    ndiag10=[ndiag10]
    ndiag11=[ndiag11]

    #FUNCION DIBUJO DE LÍNEAS----------------------------
    def dibujar(matriz,color):
        fila=0
        limfila=len(matriz)
        for fila in range(0,limfila):
            x1, y1, x2, y2 = matriz[fila]
            x1=int(x1)
            y1=int(y1)
            x2=int(x2)
            y2=int(y2)
            cv2.line(img, (x1,y1), (x2,y2), color, 3, cv2.LINE_AA)

    #APARTADO DE DIBUJO DE LINEAS DEL BLANCO Y NEGRO-----------
    verde=0,255,0
    rojo=0,0,255
    azul=255,0,0
    cian=255,255,0

    if len(fuga_todos)==0:
        #DIBUJAR LINEA SUELO
        dibujar(ones1,verde)
        #DIBUJAR LINEA TECHO
        dibujar(ones2,rojo)
        #DIBUJAR LINEA PARED IZQ
        dibujar(ones3,azul)
        #DIBUJAR LINEA PARED DCHA
        dibujar(ones4,azul)
        #DIBUJAR DIAGONAL SUPERIOR IZQUIERDA
        dibujar(ones5,cian)
        #DIBUJAR DIAGONAL INFERIOR DERECHA
        dibujar(ones6,cian)
        #DIBUJAR DIAGONAL INFERIOR IZQUIERDA
        dibujar(ones7,cian)
        #DIBUJAR DIAGONAL SUPERIOR DERECHA
        dibujar(ones8,cian)

        #APARTADO DE DIBUJO DE LINEAS DEL COLOR--------------
        ##DIBUJAR LINEA SUELO
        dibujar(ones9,verde)
        #DIBUJAR LINEA TECHO
        dibujar(ones10,rojo)
```

```
        #DIBUJAR LINEA PARED IZQ
731     dibujar(ones11,azul)
        #DIBUJAR LINEA PARED DCHA
733     dibujar(ones12,azul)
        #DIBUJAR DIAGONAL SUPERIOR IZQUIERDA
735     dibujar(ones13,cian)
        #DIBUJAR DIAGONAL INFERIOR DERECHA
737     dibujar(ones14,cian)
        #DIBUJAR DIAGONAL INFERIOR IZQUIERDA
739     dibujar(ones15,cian)
        #DIBUJAR DIAGONAL SUPERIOR DERECHA
741     dibujar(ones16,cian)

743 #DIBUJO SOLUCION FINAL-----------------------------------
    if len(fuga_todos)!=0:
745     negro=0,0,0
        dibujar(sueloprueba,negro)
747     dibujar(techoprueba,negro)
        dibujar(paredizquierda,negro)
749     dibujar(paredderecha,negro)
        #nuevas diagonales dibujadas
751     dibujar(ndiag1,negro)
        dibujar(ndiag2,negro)
753     dibujar(ndiag3,negro)
        dibujar(ndiag4,negro)
755     amarillo=0,200,200
        dibujar(sueloprueba2,amarillo)
757     dibujar(techoprueba2,amarillo)
        dibujar(paredizquierda2,amarillo)
759     dibujar(paredderecha2,amarillo)
        #nuevas diagonales dibujadas
761     dibujar(ndiag8,amarillo)
        dibujar(ndiag9,amarillo)
763     dibujar(ndiag10,amarillo)
        dibujar(ndiag11,amarillo)
765
    if destruir==1:
767     cv2.destroyAllWindows()

769 #SUPERPONER IMAGEN-----------------------
    if len(fuga_todos) != 0:
771     #se carga la textura que será empleada en la pared frontal
        textura= cv2.imread("/home/pi/Desktop/imagenestfg/paredrara.jpg")
773     textura=cv2.resize(textura,(800,800))
        texturabase=textura
775     #se visualizan las dos opciones delimitadas
        cv2.imshow("lineas", img)
777     cv2.waitKey(0)
        #el usuario escoge el recuadro en el que se introduce la imagen
779     dato = int(input("0 para recuadro negro, 1 para recuadro amarillo: "))
```

```
        #print("Has insertado ", dato)
781     color=dato

783     def mascarah(linea,techo):
            x1,y1,x2,y2=linea[0]
785         x1=int(x1)
            x2=int(x2)
787         y1=int(y1)
            y2=int(y2)
789         if x2==x1:
                x2=x2+1
791         m=(y2-y1)/(x2-x1)
            m=round(m/0.005)*0.005
793         ptoinicio=y1-m*x1
            ptoinicio=round(ptoinicio)
795         #se pinta de negro todo píxel exterior al recuadro
            for i in range(0,800):
797
                for j in range(0,800):
799                 if techo==1:
                        if j<m*i+ptoinicio:
801                         textura[j][i]=(0,0,0)
                    if techo==0:
803                     if j>m*i+ptoinicio:
                            textura[j][i]=(0,0,0)
805     def mascarav(linea,izquierda):
            y1,x1,y2,x2=linea[0]
807         x1=int(x1)
            x2=int(x2)
809         y1=int(y1)
            y2=int(y2)
811         if x2==x1:
                x2=x2+1
813         m=(y2-y1)/(x2-x1)
            m=round(m/0.005)*0.005
815         ptoinicio=y1-m*x1
            ptoinicio=round(ptoinicio)
817         for i in range(0,800):
                for j in range(0,800):
819                 if izquierda==1:
                        if j<m*i+ptoinicio:
821                         textura[i][j]=(0,0,0)
                    elif izquierda==0:
823                     if j>m*i+ptoinicio:
                            textura[i][j]=(0,0,0)
825     if color==0:
            mascarah(techoprueba,1)
827         mascarah(sueloprueba,0)
            mascarav(paredizquierda,1)
829         mascarav(paredderecha,0)
```

```
        if color==1:
831         mascarah(techoprueba2,1)
            mascarah(sueloprueba2,0)
833         mascarav(paredizquierda2,1)
            mascarav(paredderecha2,0)
835
        #se carga la imagen de la pared
837     frente = texturabase
        #se carga la imagen de la habitación
839     fondo = img
        #se carga la máscara creada
841     mascara= textura
        #se aplica la máscara
843     frente = frente.astype(float)
        fondo = fondo.astype(float)
845     mascara = mascara.astype(float)/255
        fondo = cv2.multiply(1.0 - mascara, fondo)
847     frente = cv2.multiply(mascara, frente)
        outImage = cv2.add(frente, fondo)
849     #se muestra el resultado final
        cv2.imshow("outImg", outImage/255)
851     #se mantiene la imagen unos segundos
        cv2.waitKey(5000)
853     time.sleep(2)

855 cv2.destroyAllWindows()
```

Código completo del programa de realidad aumentada

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

PLANOS

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

En este trabajo no aplica una sección de planos

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

PLIEGO DE CONDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

En este trabajo no aplica un pliego de condiciones

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

MEDICIONES

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

Índice del documento Mediciones

12 MEDICIONES

12.1. Materiales

Listado de materiales	
Materiales	Unidades
Raspberry Pi 3 modelo B	1
Cámara 5 MP-1080p	1

Tabla 12.1 – Listado de materiales necesarios para desarrollar el proyecto

12.2. Mano de obra

Mano de obra		
Tarea	Personal	Horas
Búsqueda de documentación	Graduado en Ingeniería Electrónica Industrial y Automática	30
Estudio de la documentación	Graduado en Ingeniería Electrónica Industrial y Automática	40
Análisis de soluciones	Graduado en Ingeniería Electrónica Industrial y Automática	60
Programación final	Graduado en Ingeniería Electrónica Industrial y Automática	90
Implementación del sistema de realidad aumentada	Graduado en Ingeniería Electrónica Industrial y Automática	40
Edición y redacción del documento	Graduado en Ingeniería Electrónica Industrial y Automática	50
Trabajo completo	Graduado en Ingeniería Electrónica Industrial y Automática	310

Tabla 12.2 – Personal y horas dedicadas a cada tarea

TÍTULO: **Implementación de un sistema de reconocimiento y delimitación de superficies.**

PRESUPUESTO

PETICIONARIO: **ESCUELA UNIVERSITARIA POLITÉCNICA**

AVDA. 19 DE FEBREIRO, S/N

15405 - FERROL

FECHA: JUNIO DE 2020

AUTOR: EL ALUMNO

Fdo.: MANUEL PALACIOS FRAGOSO

Índice del documento PRESUPUESTO

13 PRESUPUESTO

13.1. Materiales

Listado de materiales		
Materiales	Unidades	Precio
Raspberry Pi 3 modelo B	1	40,23 €
Cámara 5 MP-1080p	1	8,99 €
Total		49,22 €

Tabla 13.1 – Listado de precios de los materiales empleados

13.2. Mano de obra

Mano de obra			
Tipo de mano de obra	Número de horas	€/hora	Precio total
Graduado en Ingeniería Electrónica Industrial y Automática	310	35 €	10.850 €

Tabla 13.2 – Coste la mano de obra empleada en el proyecto

13.3. Coste total

Coste total del proyecto	
Concepto	Precio
Materiales	49,22 €
Mano de obra	10.850 €
Total	10.899,22 €

Tabla 13.3 – Coste total del proyecto